Static analysis of XML security views and query rewriting

Static Analysis of XML Security Views and Query Rewriting[☆]

Benoît Groz[a,b,d], Sławomir Staworko[a,c], Anne-Cecile Caron[a,b], Yves Roos[a,b], Sophie Tison[a,b]

[a] *Mostrare project, INRIA Lille Nord-Europe & LIFL (CNRS UMR8022)*
[b] *University of Lille 1*
[c] *University of Lille 3*
[d] *ENS Cachan*

Abstract

In this paper, we revisit the view based security framework for XML without imposing any of the previously considered restrictions on the class of queries, the class of DTDs, and the type of annotations used to define the view. First, we study *query rewriting* with views when the classes used to defined queries and views are Regular XPath and MSO. Next, we investigate problems of *static analysis* of security access specifications (SAS): we introduce the novel class of *interval-bounded* SAS and we define three different manners to compare views (i.e. on queries), with a security point of view. We provide a systematic study of the complexity for deciding these three comparisons, when the depth of the XML documents is bounded, when the document may have an arbitrary depth but the queries defining the views are restricted to guarantee the interval-bounded property, and in the general setting without restriction on queries and document.

Keywords: XML, security views, query rewriting, determinacy.

1. Introduction

The wide acceptance of XML as the format for data representation and exchange clearly demonstrates the need for a general and flexible framework of secure access for XML databases. While security specification and enforcement are well established in relational databases, their methods and approaches cannot be easily adapted to XML databases. This is because an XML document stores information not only in its data nodes but also in the way it is structured. Consequently, the problem of secure access to XML databases has its own particular flavor and requires dedicated solutions.

[☆]This work is partially supported by the INRIA collaboration program (Actions de Recherches Collaboratives de l'INRIA)

The *view-based security framework* for XML databases [29] has received an increased attention from both the theoretical and practical angle [12, 19, 13, 27, 34, 28, 14]. It can be summarized as follows:

- The administrator provides the schema of the document together with the *security access specification* (SAS) defining nodes accessible by the user.
- A *virtual view* comprising all accessible nodes is defined; the view is never materialized but the user is given some knowledge of its schema.
- Every query over the view is *rewritten* to an equivalent query over the underlying document and then evaluated.

The view-based security framework is parametrized by the class of queries, typically a fragment of XPath, and the type of formalism used to define the schema with the security access specification, typically an annotated DTD. Previous research often imposed various restrictions on these two parameters in order to facilitate the tasks relevant to the framework. For instance, taking the class of downward XPath queries allows to use the knowledge of the document DTD to benefit the query rewriting [12]. The task can be further simplified if the node accessibility is *downward closed* i.e., all descendants of an inaccessible node are inaccessible as well [2]. For similar reasons, in some works only non-recursive DTDs are considered [12, 27].

The restrictions may easily limit the versatility of the framework and in this paper we revisit the framework and take two large classes of queries: Regular XPath queries ($\mathcal{X}Reg$) and Monadic Second-order Logic queries (MSO) represented with tree automata. The two formalisms are also used to define accessibility of nodes in the source document. Usually, the schema is assumed to be specified with a DTD. In this paper, the schema could also be described by richer languages, e.g. extended DTDs.

In the first part of the paper, we revisit the problem of rewriting queries over views. Recall that in the case of the standard XPath queries [12], there are queries that cannot be rewritten, because the language is not powerful enough to capture node accessibility, and consequently, various restrictions need to be employed. Our work shows that both $\mathcal{X}Reg$ and MSO enjoy the closure on rewriting under views. In both cases, the rewritings are quadratic (combined complexity including both the size of the input query and the security access specification).

In the second part of the paper, we study the problem of comparing two security access specifications. This problem is best motivated in situations where the administrator changes the specification of accessible nodes, for instance restricts access to some nodes, and would like to obtain some guarantees that no information has been inadvertently released with the change.

We consider three kinds of comparisons of SAS and investigate their computational implications. The first comparison considers only the accessibility of nodes in a document. Comparing SAS from this perspective is essentially testing the containment of queries used to specify accessible nodes, a problem known to be EXPTIME-complete for both $\mathcal{X}Reg$ and tree automata.

Paradoxically, when we restrict the access (i.e. the set of accessible nodes) we can make some new information available about the accessible nodes. E.g., let us suppose that user A sees all the patients of a hospital and user B sees only the patients of service X. For the first comparison, user B has a more restrictive view than user A. However, B gets some information that A does not get: A will a priori be unable to distinguish patients of the service X from the others. In order to capture this phenomenon, the second comparison identifies and compares the information that can be obtained on the underlying document. This information is defined as the set of all queries on the underlying document that can be expressed as queries over the view.

In a nutshell, A will have a more restrictive access than B for that comparison if every query computed by A can be simulated by B. We prove that this can be expressed by "View A can been considered as a view on the view B". So, this notion is related to single-view query rewriting [7, 10] as well as to composition of views [2].

We prove easily that this second comparison refines the first one. However, both can be considered too strong in some sense. Indeed, even when the view of user A selects nodes outside the view of B, it may still be possible to "reconstruct" the view of A from the view of B with some knowledge from the schema. Consider for instance a list of entries of the form `entry(name, phone number)`. Then `entry` is just syntactic sugar: the list of pairs `(name, phone number)` contains the same information as the list of entries `entry(name, phone number)`. So deleting the `entry` keeps information intact. The third comparison takes into account the possibility to deduce information on the hidden part from the SAS. For this, the third comparison uses all data, visible or not, based on the following idea: view A is more restrictive than view B if every (boolean) information about the source that is certain for A is also certain for B. In other words, the definition is based on the well known notion of *certain answers* and it can be related to view-based query answering. This can be also related to another approach to guarantee privacy [21], where the administrator defines the information he considers secret by using a boolean query $\mathcal{Q}$. In this context, being more restrictive can be considered as "keeping more secrets".

The second and third comparisons are very powerful and not surprisingly they turn out to be undecidable in general. Consequently, we introduce a novel class of *interval-bounded* SAS for which those problems become tractable. Interval-bounded SAS generalize both non-recursive views, and downward closed access specifications: in interval-bounded SAS an inaccessible node may have accessible descendants as long as the number of consecutive inaccessible descendants is bounded by some constant (independent from the document).

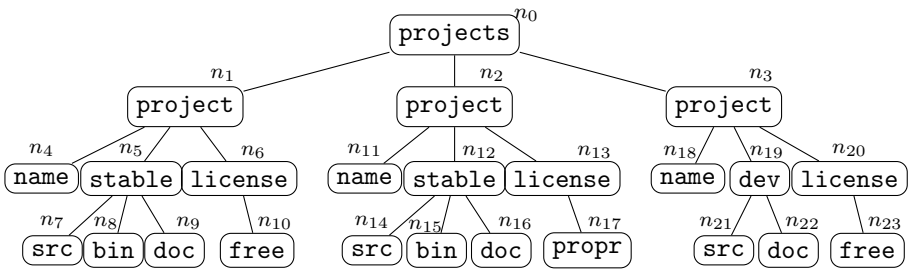
2. Preliminaries

XML Documents.

We assume a finite set of node labels Σ and model XML documents with unranked ordered labeled trees. Formally, a Σ-tree is a finite structure $t =$

$(N_t, root_t, child_t, next_t, \lambda_t)$, where N_t is a set of nodes, $root_t \in N_t$ is a distinguished root node, $child_t \subseteq N_t \times N_t$ is the parent-child relation, $next_t \subseteq N_t \times N_t$ is the next-sibling relation, and $\lambda_t : N_t \to \Sigma$ is the function assigning to every node its label. The set of all Σ-trees is denoted by T_Σ. We remark that we *do not assume* the set of nodes to be a prefix closed subset of $\mathbb{N}^*$, owing to our construction of view trees; we shall later define view trees obtained by removing nodes from some tree t and preserving the others: even when the original trees use a prefix-closed subset of identifiers, the resulting view trees need not. Moreover, equality of trees should not be confused with isomorphism: two trees are equal if and only if all the elements of their underlying structure are the same, including the node set.

Example 1. Figure 1 contains an example of a tree representing an XML database with information on software development projects. Every project



$$N_{t_0} = \{n_0, n_1, n_2, \ldots\}, \quad root_{t_0} = n_0, \quad \lambda_{t_0} = \{(n_0, \texttt{projects}), (n_1, \texttt{project}), \ldots\},$$
$$child_{t_0} = \{(n_0, n_1), (n_0, n_2), \ldots\}, \qquad next_{t_0} = \{(n_1, n_2), (n_2, n_3), (n_4, n_5), \ldots\}.$$

Figure 1: Tree t_0.

has a name and a type of license (either free or proprietary). Projects under development come with their source codes and documentation, whereas stable projects have also binaries.

Queries and Annotations

A *query* Q is a mapping from T_Σ to $\bigcup_{t \in T_\Sigma} \mathcal{P}(N_t)$ which satisfies that for each t in T_Σ, $Q(t)$ is included in N_t. The set of nodes $Q(t)$ is called the set of *answers* of the query Q on the tree t. The *domain* $\mathrm{dom}(Q)$ is the set of trees t in T_Σ such that $Q(t)$ is not empty. A query is *root preserving* if for all t in T_Σ, either $Q(t)$ is empty, or $Q(t)$ contains (at least) the root of t.

In this article, we only consider queries that are *closed by isomorphism*: a query Q is said closed by isomorphism if for all trees t and t' and all isomorphism ϕ such that $t' = \phi(t)$, $Q(t') = \phi(Q(t))$. As a consequence, the domains of these queries are also closed by isomorphism.

An *annotation* A is a mapping from T_Σ to $T_{\Sigma\times\{\mathbb{0},\mathbb{1}\}}$ such that $A(t)$ is a relabeling of t, replacing the label $\lambda_t(n)$ of each node $n \in N_t$ by some $(\lambda_t(n), i)$ where $i \in \{\mathbb{0},\mathbb{1}\}$ is the annotation of node n that we denote by $A(n)$.

Given a query Q, we denote by A_Q the annotation such that $\forall t \in T_\Sigma$, $\forall n \in N_t$ $A_Q(n) = \mathbb{1}$ *iff* $n \in Q(t)$. An annotation A_Q is *root preserving* if Q is root preserving.

Given a tree t in $T_{\Sigma\times\{\mathbb{0},\mathbb{1}\}}$, we will denote by $\Pi_\Sigma(t)$ the relabeling of t replacing for each node of t its label $(\alpha, i) \in \Sigma \times \{\mathbb{0},\mathbb{1}\}$ with α. We say that a language $L \subseteq T_{\Sigma\times\{\mathbb{0},\mathbb{1}\}}$ is *maximal* if, for all non-isomorphic trees t and t' in L, it holds that $\Pi_\Sigma(t) \not\sim \Pi_\Sigma(t')$. By definition, for any annotation A, the language $A(T_\Sigma)$ is maximal. Maximal languages allow to represent the result of a query Q over some tree t within a single tree: the unique tree t' in the language such that $\Pi_\Sigma(t') = t$. Of course other representations could be considered but that one is quite convenient for view-based reasoning.

Security Views.

In a security framework, we want to hide some nodes of a document. Queries and annotations provide a simple model for security policy. The *(security) view* defined by a root-preserving query Q maps every document $t \in \mathrm{dom}(Q)$ to $\mathcal{V}iew\,(Q,t)$, the *view document* obtained from t by removing all the nodes that are not selected by Q. Those are equivalently the nodes labeled with $\mathbb{0}$ by the annotation A_Q. Removing a node causes its children to be *adopted* by (or *linked* to) the parent of the node. We assume that a query defining a security policy is always root preserving in order to guarantee that view documents are trees. An example of such a view is given in Fig. 3 in the case of security views defined by *annotated DTDs* that are a practical and simple way to define security policies. We extend our notation to languages, denoting by $\mathcal{V}iew\,(Q,L)$ the set $\bigcup_{t\in L} \mathcal{V}iew\,(Q,t)$, for any tree language $L \subseteq \mathrm{dom}(Q)$.

Regular XPath queries.

A standard manner of expressing queries is to use XPath expressions. In particular, the class $\mathcal{X}Reg$ of Regular XPath expressions [23] over Σ-trees is defined by the following grammar (with a varying over Σ and $\mathcal{X}$ being the starting symbol):

$$\begin{aligned}
\alpha &::\equiv \mathsf{self} \mid \Downarrow \mid \Uparrow \mid \Rightarrow \mid \Leftarrow \\
f &::\equiv \mathsf{self}{::}a \mid \chi \mid \mathsf{true} \mid \mathsf{false} \mid \mathsf{not}\ f \mid f\ \mathsf{and}\ f \mid f\ \mathsf{or}\ f \\
\mathcal{X} &::\equiv \alpha \mid [f] \mid \mathcal{X}/\mathcal{X} \mid \mathcal{X} \cup \mathcal{X} \mid \mathcal{X}^*
\end{aligned}$$

Essentially, a $\mathcal{X}Reg$ expression is a regular expression of base axes and filter expressions. Filter expressions are Boolean combinations of node label tests and existential path tests. We define several macros: α^+ is short for α^*/α, $\mathcal{X}[f]$ is $\mathcal{X}/[f]$, $\alpha{::}a$ stands for $\alpha[\mathsf{self}{::}a]$, and $\alpha{::}*$ is simply α, where a is a symbol, $\mathcal{X}$ a $\mathcal{X}Reg$ expression, f a filter expression, and α a base axis or its closure. The semantics of $\mathcal{X}Reg$ is defined in Fig. 2 (Boolean connectives are

$$
\begin{aligned}
[\![\mathsf{self}]\!]_t &= \{(n,n) \mid n \in N_t\}, & [\![\mathcal{X}_1/\mathcal{X}_2]\!]_t &= [\![\mathcal{X}_1]\!]_t \circ [\![\mathcal{X}_2]\!]_t,\\
[\![\Downarrow]\!]_t &= child_t, & [\![\mathcal{X}_1 \cup \mathcal{X}_2]\!]_t &= [\![\mathcal{X}_1]\!]_t \cup [\![\mathcal{X}_2]\!]_t,\\
[\![\Uparrow]\!]_t &= child_t^{-1}, & [\![\mathcal{X}^*]\!]_t &= [\![\mathcal{X}]\!]_t^*,\\
[\![\Rightarrow]\!]_t &= next_t, & [\![[f]]\!]_t &= \{(n,n) \in N_t \mid (t,n) \models f\}\\
[\![\Leftarrow]\!]_t &= next_t^{-1}, & (t,n) &\models \mathsf{self}::a \text{ iff } \lambda_t(n) = a,\\
& & (t,n) &\models \mathcal{X} \text{ iff } \exists n' \in N_t.\, (n,n') \in [\![\mathcal{X}]\!]_t.
\end{aligned}
$$

Figure 2: The semantics of $\mathcal{X}Reg$.

interpreted in the usual manner). For an expression $\mathcal{X}$ in $\mathcal{X}Reg$, $[\![\mathcal{X}]\!]_t$ is the binary reachability relation on the nodes of t defined by the expression $\mathcal{X}$. By $(t,n) \models f$ we denote that the filter f is *satisfied* at the node n of the tree t. We say that an expression $\mathcal{X}$ is *satisfied* in the tree t if $(t, root_t) \models \mathcal{X}$. Then an expression $\mathcal{X}$ in $\mathcal{X}Reg$ defines a query $Q_{\mathcal{X}}$ where the set of answers to the query $Q_{\mathcal{X}}$ in a tree t is defined as

$$Q_{\mathcal{X}}(t) = \{n \in N_t \mid (root_t, n) \in [\![\mathcal{X}]\!]_t\}.$$

For instance, $\Downarrow$::`project`[$\Downarrow$::`stable`]/$\Downarrow$::`name` defines a query Q_0 that selects (the nodes storing) the names of all stable projects. The set of answers to Q_0 in t_0 (Fig. 1) is $Q_0(t_0) = \{n_4, n_{11}\}$.

We recall from [23] that $\mathcal{X}Reg$ is closed under inversion, i.e. for every expression $\mathcal{X}$ there exists an expression $\mathcal{X}^{-1}$ such that $[\![\mathcal{X}^{-1}]\!]_t = [\![\mathcal{X}]\!]_t^{-1}$ for any tree t. Basically, $\mathcal{X}^{-1}$ is obtained by reversing the base axes and the order of composition on the top most level (filter expressions are unchanged). Naturally, $|\mathcal{X}^{-1}| = |\mathcal{X}|$.

Annotated DTDs.

A *Document Type Definition* (DTD) over Σ is a triple $D = (\Sigma, r, P)$ where $r \in \Sigma$ and P is a function that maps Σ to regular expressions over Σ. We allow regular expressions defined with the grammar

$$E ::\equiv \mathsf{empty} \mid a \mid E\ \text{','}\ E \mid E\ \text{'|'}\ E \mid E\text{'*'}$$

where empty defines the empty sequence, a is a symbol of Σ, E, E is the concatenation, $E|E$ is the union, and E^* is the Kleene closure. In the sequel, we present DTDs using rules of the form $a \to E$ and if for a symbol a the rule is not specified, then $a \to \mathsf{empty}$ is implicitly assumed. The dependency graph of a DTD $D = (\Sigma, r, P)$ is a directed graph whose node set is Σ and the set of edges contains (a,b) if $P(a)$ uses the symbol b. A DTD is *recursive* iff its dependency graph is cyclic. The size $|D|$ of a DTD $D = (\Sigma, r, P)$ is the sum of the sizes of the regular expressions $P(\alpha)$ appearing in D.

A Σ-tree t *satisfies* a DTD $D = (\Sigma, r, P)$ if for every natural k and every node n having exactly k children $n_1, \ldots, n_k$ (listed in the document order), we

have $\lambda_t(n_1)\cdots\lambda_t(n_k) \in L(P(\lambda_t(n)))$. By $L(D)$ we denote the set of all Σ-trees that satisfy D.

In [29], a security view is defined from a DTD specifying nodes accessible by the user. This framework has been widely studied from both the theoretical and practical angle [12, 19, 13, 27, 34, 28, 14].

In this framework, an *annotated DTD* (D, X) consists of a DTD D and an *access function* X. This access function specifies the accessibility of document nodes. Formally, an *access function* is given by a (possibly partial) function X that maps $\Sigma \times \Sigma$ to $\mathcal{X}Reg$ filter expressions. Its size $|X|$ is simply the sum of the sizes of all filter expressions used in X. The function X defines the security access function of nodes as follows. A node n labelled with b whose parent is labelled with a is *accessible* w.r.t. X if the filter expression $X(a, b)$ is satisfied at the node n. If $X(a, b)$ is not defined, then accessibility of the parent is used (inheritance). Finally, the root node of any tree validating the DTD D is always accessible. Thus, we can associate with each annotated DTD (D, X) a root preserving query $Q_{(D,X)}$ with domain $L(D)$ which maps every tree $t \in L(D)$ to the set of its accessible nodes. For this reason, annotated DTD are used to define security views. The size $|A|$ of the annotated DTD $A = (D, X)$, is $|D| + |X|$.

Example 2. The DTD D_0 below captures the schema of XML databases described in Example 1. We define here the annotated DTD $A_0 = (D_0, X_0)$.

$\texttt{projects} \rightarrow \texttt{project}^*$
$\texttt{project} \rightarrow \texttt{name}, (\texttt{stable} \mid \texttt{dev}), \texttt{license}$
$X_0(\texttt{project}, \texttt{stable}) = \texttt{false}$
$X_0(\texttt{project}, \texttt{dev}) = \texttt{false}$
$\texttt{license} \rightarrow \texttt{free} \mid \texttt{propr}$
$\texttt{stable} \rightarrow \texttt{src}, \texttt{bin}, \texttt{doc}$
$X_0(\texttt{stable}, \texttt{src}) = [\Uparrow^*::\texttt{project}/\Downarrow^*::\texttt{free}]$
$X_0(\texttt{stable}, \texttt{doc}) = \texttt{true}$
$\texttt{dev} \rightarrow \texttt{src}, \texttt{doc}$
$X_0(\texttt{dev}, \texttt{src}) = [\Uparrow^*::\texttt{project}/\Downarrow^*::\texttt{free}]$
$X_0(\texttt{dev}, \texttt{doc}) = \texttt{true}$

The access function X_0 gives access to all projects but in return hides the information whether or not the project is stable (in particular, it hides binaries). Additionally, X_0 hides the source code of all projects developed under proprietary license.

In the tree t_0 from Fig. 1 the root node `projects` is accessible and all nodes `project` are accessible by inheritance. The nodes `name` and `license` with their children are accessible by inheritance as well. X_0 implicitly states that `stable` and `dev` are not accessible, and the nodes `bin` are inaccessible by inheritance. On the other hand, X_0 overrides the inheritance for nodes `doc` and makes them accessible. Finally, the accessibility of `src` nodes is conditional: only n_7 and n_{21} are accessible because only those satisfy the specified conditions, $X_0(\texttt{stable}, \texttt{src})$ and $X_0(\texttt{dev}, \texttt{src})$ resp. Figure 3 presents $\mathcal{V}iew\,(Q_{A_0}, t_0)$ for t_0 from Fig. 1.

The following lemma allows us to translate an access function into a $\mathcal{X}Reg$ filter:

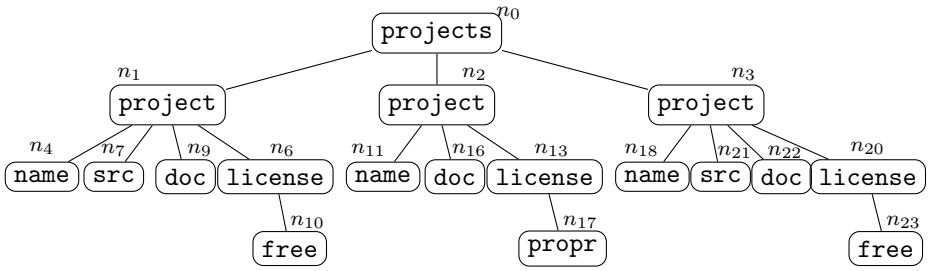


Figure 3: The view $\mathcal{V}iew\,(Q_{A_0}, t_0)$.

Lemma 1. *For any access function X there exists a $\mathcal{X}Reg$ filter expression $\mathcal{X}^X_{\text{acc}}$ such that for any tree $t \in T_\Sigma$, a node $n \in N_t$ is accessible in t w.r.t. X if and only if $(t, n) \models \mathcal{X}^X_{\text{acc}}(t)$. Moreover, $\mathcal{X}^X_{\text{acc}}$ can be constructed in $O(|X|)$ time.*

Proof. By $dom(X)$ we denote the set of pairs of symbols for which X is defined. We begin by defining two filter expressions. The first checks if X defines a filter expression for the current node

$$\mathcal{X}_{\text{dom}} \coloneqq \bigvee_{(a,b)\in dom(X)} \big(\mathsf{self}::b \text{ and } \Uparrow::a\big),$$

and if it is the case, the second filter expression is used to evaluate it

$$\mathcal{X}_{\text{eval}} \coloneqq \bigvee_{(a,b)\in dom(X)} \big(\mathsf{self}::b \text{ and } \Uparrow::a \text{ and } X(a,b)\big).$$

Finally, we restate the definition of accessibility using $\mathcal{X}Reg$

$$\mathcal{X}^X_{\text{acc}} \coloneqq ([\text{not } \mathcal{X}_{\text{dom}}]/\Uparrow)^*/[\text{not}(\Uparrow) \text{ or } \mathcal{X}_{\text{eval}}]. \qquad \square$$

The $\mathcal{X}Reg$ expression $\mathcal{X}^X_{\text{acc}}$ associated with an access function X is a filter that can be applied on every node in order to check its visibility. The $\mathcal{X}Reg$ expression $\Downarrow^*[\mathcal{X}^X_{\text{acc}}]$ therefore selects the nodes that are visible for access function X. In the following, given an access function X, we will denote the query $Q_{\Downarrow^*[\mathcal{X}^X_{\text{acc}}]}$ simply by Q_X.

From [23], we know that a DTD can effectively be transformed into an equivalent $\mathcal{X}Reg$ expression in linear time. From this result and Lemma 1 we get the following lemma:

Lemma 2. *For any annotated DTD $A = (D, X)$ the query $Q_{(D,X)}$ can be defined in $\mathcal{X}Reg$. This means we can compute a $\mathcal{X}Reg$ query Q_A with domain $L(D)$ such that for any tree $t \in L(D)$, a node $n \in N_t$ is accessible in t w.r.t. X if and only if $n \in Q_A(t)$. Moreover, Q_A can be constructed in $O(|A|)$ time.*

Thanks to Lemma 2, in the case when trees have to be validated against some DTD D and the visibility of the nodes is given by a query V, we assume in the following that a single query $Q_{(D,V)}$ is used to define both validation of trees and visibility of the nodes.

Queries and automata

Instead of $\mathcal{X}Reg$ formulas we could alternatively use monadic second order logic (MSO) formulas to represent queries. An MSO formula with n free first-order variables, interpreted over unranked ordered labeled trees, defines a query that selects tuples of nodes and it is known from [32, 31] that MSO queries are strictly more expressive than $\mathcal{X}Reg$ queries.

In the sequel, instead of MSO formulas, we shall directly use automata since it is well-known from [33] that the class of regular ranked tree languages is exactly the class of MSO-definable ranked tree languages and from [11] that this equivalence also holds in the unranked case. Several classes of (unranked) tree automata for XML with the expressive power of *MSO* have been recently studied. Each could be used in order to define queries in the following way: to any *MSO* formula ϕ with one free first-order variable we associate a query Q_ϕ then an annotation A_{Q_ϕ}. The language $A_{Q_\phi}(T_\Sigma)$ is a regular language over $T_{\Sigma\times\{\mathbb{0},\mathbb{1}\}}$ and is therefore recognized by some automaton $\mathcal{A}_\phi$ (in the class of automata that has been chosen). Conversely, any automaton $\mathcal{A}$ over $T_{\Sigma\times\{\mathbb{0},\mathbb{1}\}}$ that recognizes a maximal language $L(\mathcal{A})$ is associated with a query $Q_\mathcal{A}$ defined by:

- $Q_\mathcal{A}(t) = \varnothing$ for every tree t that is not in $\Pi_\Sigma(L(\mathcal{A}))$
- for every tree t in $\Pi_\Sigma(L(\mathcal{A}))$, there is a unique $t' \in L(\mathcal{A})$ that satisfies $t = \Pi_\Sigma(t')$. Then $Q_\mathcal{A}(t)$ is the set of all nodes of t' with label in $\Sigma \times \{\mathbb{1}\}$.

We extend the notion of root preservation to automata: an automaton $\mathcal{A}$ is root preserving if the query $Q_\mathcal{A}$ is root preserving.

In this paper, we use the class of Visibly Pushdown Automata (VPA). Visibly pushdown automata (VPA) have been introduced by Rajeev Alur and Parthasarathy Madhusudan in [1] in order to model program analysis. VPA are special pushdown (word) automata whose stack behavior is driven by the input symbol according to a partition of the alphabet. Although they were not initially defined for this purpose, VPA are very useful for processing XML streams, since they can recognize well-matched languages defined over an input alphabet of opening tags and closing tags.

Let Σ an alphabet. We denote by $\hat{\Sigma} = \{\mathrm{op}, \mathrm{cl}\} \times \Sigma$ the corresponding tag alphabet, where for any label $a \in \Sigma$, (op, a) is an *opening* a and (cl, a) is a *closing* a. Then for any tree $t \in T_\Sigma$ we can define its linearization as usual by: $\mathrm{lin}(a(t_1, \cdots, t_n)) = (\mathrm{op}, a)\,\mathrm{lin}(t_1)\cdots\mathrm{lin}(t_n)\,(\mathrm{cl}, a)$. So, there is a bijection between the nodes of t and pairs of corresponding opening and closing tag. We extend this notation to tree languages: $\forall L \subseteq T_\Sigma, \mathrm{lin}(L) = \bigcup_{t\in L} \mathrm{lin}(t)$.

Since a tree language L is regular if and only if $\mathrm{lin}(L)$ is recognized by some visibly pushdown automaton ([1]), these automata provide a suitable formalism for representing MSO-definable queries. Let us define formally visibly pushdown automata.

Definition 1. A *visibly pushdown automaton* over an alphabet Σ is a tuple $\mathcal{A} = (\Sigma, S, \Gamma, I, F, R)$ where

- Σ is the *input alphabet*,
- S is a finite set of states,
- Γ is a finite alphabet of *stack symbols*,
- $I \subseteq S$ is the set of *initial states*,
- $F \subseteq S$ is the set of *final states*,
- and $R \subseteq S \times \{\text{op}, \text{cl}\} \times \Sigma \times \Gamma \times S$ is the set of *rules*.

The *size* of $\mathcal{A}$ is $|S| + |\Gamma| + |\Delta|$. A rule $(q, \iota, a, \gamma, q') \in \Delta$ is written $q \xrightarrow{(\iota,a):\gamma} q'$. When ι is equal to op, then $q \xrightarrow{(\text{op},a):\gamma} q'$ is a *push rule*. It means that *if the current state is q and the input letter is an opening a then one can push γ into the stack and set the current state to q'*. Symmetrically, a rule $q \xrightarrow{(\text{cl},a):\gamma} q'$ is a *pop rule*. It means that *if the current state is q and the top of the stack is γ and the input letter is a closing a then one can pop γ from the stack and set the current state to q'*.

We will sometimes define VPAs with ϵ-transitions of the form (q, ϵ, q') with $q, q' \in \Sigma$ in the rules. This does not increase the expressiveness of the VPAs because the ϵ-transitions can be eliminated in polynomial time. To eliminate the ϵ-transitions we can add a new rule $(q_0, \iota, a, \gamma, q'_k)$ in Δ for every $(q, \iota, a, \gamma, q') \in \Delta$ and every $j, k \leqslant |S|$, $q_0, q_1, \ldots, q_j \in S$ and $q' = q'_0, \ldots q'_k \in S$ satisfying the following three conditions: (1) $q_j = q$, (2) for every $i < j$, $(q_i, \epsilon, q_{i+1}) \in \Delta$, and (3) for every $i < k$, $(q'_i, \epsilon, q'_{i+1}) \in \Delta$.

Let $\mathcal{A} = (\Sigma, S, \Gamma, I, F, R)$ be a visibly pushdown automaton, then a *run* of $\mathcal{A}$ from q_0 to q_m over a word $w = a_1 a_2 \ldots a_m \in (\{\text{op}, \text{cl}\} \times \Sigma)^*$ is a sequence $(q_0, \sigma_0), (q_1, \sigma_1), \ldots (q_m, \sigma_m)$ with $q_i \in S$ and $\sigma_i \in \Gamma^*$ for every $i \in \{0, \ldots m\}$, such that $\sigma_0 = \sigma_m = \varepsilon$ and for every $i < m$, there are some $b \in \Sigma$ and $\gamma \in \Gamma$ such that either $a_i = (\text{op}, b)$, $(q_i, \text{op}, b, \gamma, q_{i+1}) \in R$ and $\sigma_{i+1} = \sigma_i \cdot \gamma$, or otherwise $a_i = (\text{cl}, b)$, $(q_i, \text{cl}, b, \gamma, q_{i+1}) \in R$ and $\sigma_i = \sigma_{i+1} \cdot \gamma$. The run is *accepting* if $q_0 \in I$ and $q_m \in F$. By extension, a run of $\mathcal{A}$ over tree t is defined as a run of $\mathcal{A}$ over $\text{lin}(t)$. A word(resp. a tree) s is recognized by $\mathcal{A}$ if there is an accepting run of $\mathcal{A}$ over s. In this work, we only consider documents represented as trees, so the transitions of the VPA must ensure that every accepted word is the linearization of some tree: for instance, a word like $(\text{op}, a)(\text{cl}, b)$ is not accepted by any VPA. We also note that a run ρ of $\mathcal{A}$ over a tree t induces a function, which we abusively also denote by ρ, from the nodes of t to a pair of states (q_{in}, q_{out}). Given $n \in N_t$, if a_i, a_j is the pair of opening and closing tag corresponding to node n in the word $w = \text{lin}(t)$ above, then $\rho(n)$ is defined as (q_i, q_{j-1}). Note that we have $i = j - 1$ if n is a leaf of t.

We denote by $\hat{L}(\mathcal{A}) \subseteq \hat{\Sigma}^*$ the word language accepted (or recognized) by $\mathcal{A}$ and we denote by $L(\mathcal{A})$ the tree language accepted (or recognized) by $\mathcal{A}$ that is $\{t \in T_\Sigma \mid \text{lin}(t) \in \hat{L}(\mathcal{A})\}$. A *query automaton* ($\mathcal{QA}$) is a VPA $\mathcal{A}$ over alphabet $\Sigma \times \{\mathbb{0}, \mathbb{1}\}$ such that $L(\mathcal{A})$ is a maximal language.

We next write a rough pumping lemma for VPAs. Actually we distinguish a vertical and an horizontal pumping argument, which are both used in the section about policy comparison.

Lemma 3. *Let $\mathcal{A}$ a VPA, t a tree in $L(\mathcal{A})$ and ρ an accepting run of $\mathcal{A}$ on t. If there are nodes $n \neq n'$ in t with n' a descendant of n and $\rho(n) = \rho(n')$, then the tree t' also belongs to $L(\mathcal{A})$, where t' is obtained from t by replacing the subtree rooted at n (n included) by the subtree rooted at n'.*

Moreover, if there is a node n in t with children $n_1, \ldots, n_k$ such that for some $1 \leqslant i < j \leqslant k$ the property $\rho(n_i) = \rho(n_j)$ is satisfied, then the tree obtained from t by removing $n_{i+1}, \ldots, n_j$ and their descendants also belongs to $L(\mathcal{A})$.

The proof is essentially immediate from the definition of accepting runs for VPAs.

3. Query rewriting over XML views

In this section, we identify two classes of queries which are closed under query rewriting. A class $\mathcal{C}$ of queries is closed under query rewriting if and only if for any query $Q_1 \in \mathcal{C}$ and for any root preserving query $Q_2 \in \mathcal{C}$, the query $\mathsf{Rewrite}(Q_1, Q_2)$ belongs to $\mathcal{C}$, where $\mathsf{Rewrite}(Q_1, Q_2)$ is defined by $\mathsf{Rewrite}(Q_1, Q_2)(t) = Q_1(\mathcal{V}iew(Q_2, t))$ for any tree t.

3.1. Regular XPath

The rewriting technique for downward queries [12] relies on the knowledge of the DTD. Our rewriting method works independently of the DTD. The method uses the fact that accessibility of a node can be defined with a single filter expression (Lemma 1). This filter is used to construct rewritings of the base axes (Lemma 4), which are used to rewrite the user queries.

Lemma 4. *For any access function X and any $\alpha \in \{\Downarrow, \Uparrow, \Rightarrow, \Leftarrow\}$ there exists a $\mathcal{X}Reg$ expression R^X_α such that $[\![R^X_\alpha]\!]_t = [\![\alpha]\!]_{\mathcal{V}iew(Q_X,t)}$ for every tree t. Moreover, $|R^X_\alpha| = O(|X|)$.*

Proof. Essentially, the rewriting R^X_α defines paths, traversing inaccessible nodes only, from one accessible node to another accessible node in a manner consistent with the axis α. For the vertical axes the task is quite simple:

$$R^X_\Downarrow \coloneqq [\mathcal{X}^X_{\mathrm{acc}}]/\Downarrow/([\mathsf{not}\,\mathcal{X}^X_{\mathrm{acc}}]/\Downarrow)^*/[\mathcal{X}^X_{\mathrm{acc}}] \quad \text{and} \quad R^X_\Uparrow \coloneqq (R^X_\Downarrow)^{-1}$$

Rewritings of the horizontal axes are slightly more complex and we first define auxiliary filter expressions:

$$f^\exists_\downarrow \coloneqq ([\mathsf{not}\,\mathcal{X}^X_{\mathrm{acc}}]/\Downarrow)^*/[\mathcal{X}^X_{\mathrm{acc}}], \quad f^\varnothing_\downarrow \coloneqq \mathsf{not}\, f^\exists_\downarrow, \quad f^\varnothing_\rightarrow \coloneqq (\Rightarrow/[f^\varnothing_\downarrow])^*/[\mathsf{not}(\Rightarrow)].$$

$f^\exists_\downarrow$ checks that the current node or any of its descendants is accessible. Conversely, $f^\varnothing_\downarrow$ checks whether the current node and all of its descendants are inaccessible. Similarly, $f^\varnothing_\rightarrow$ verifies that only inaccessible nodes can be found among the siblings following the current node and their descendants.

The expression $R^X_{\Rightarrow}$ seeks the next accessible node among the following siblings of the current node and their descendants. However, if there are no such nodes but the parent is inaccessible, the next accessible node is sought among the following siblings of the parent. The last step is repeated recursively if needed.

$$R^X_{\Rightarrow} \coloneqq [\mathcal{X}^X_{\text{acc}}]/([f^{\varnothing}_{\rightarrow}]/\Uparrow/[\mathsf{not}\,\mathcal{X}^X_{\text{acc}}])^*/\Rightarrow/(\,[(\mathsf{not}\,\mathcal{X}^X_{\text{acc}})\ \mathsf{and}\ f^{\varnothing}_{\downarrow}]/\Rightarrow \cup \\ [(\mathsf{not}\,\mathcal{X}^X_{\text{acc}})\ \mathsf{and}\ f^{\exists}_{\downarrow}]/\Downarrow/[\neg\Leftarrow])^*/[\mathcal{X}^X_{\text{acc}}]$$

and $R^X_{\Leftarrow} \coloneqq (R^X_{\Rightarrow})^{-1}$. We observe that $|R^X_\alpha| = O(|X|)$ for every $\alpha \in \{\Downarrow, \Uparrow, \Rightarrow, \Leftarrow\}$. □

Theorem 1. *$\mathcal{X}Reg$ is closed under query rewriting. Moreover, given a $\mathcal{X}Reg$ query Q and a root preserving $\mathcal{X}Reg$ query Q', $\mathsf{Rewrite}(Q, Q')$ is computable in time $O(|Q| * |Q'|)$.*

Proof. The function $\mathsf{Rewrite}(Q, Q')$ replaces in Q every occurrence of a base axis $\alpha \in \{\Downarrow, \Uparrow, \Rightarrow, \Leftarrow\}$ with $R^{Q'}_\alpha$. A simple induction over the size of Q shows that $[\![Q]\!]_{\mathcal{V}iew(Q',t)} = [\![\mathsf{Rewrite}(Q, Q')]\!]_t$, Lemma 4 handling the nontrivial base cases. Since the root is always accessible, we get $Q\,(\mathcal{V}iew\,(Q', t)) = \mathsf{Rewrite}(Q, Q')\,(t)$. We note that the rewritten query is constructed in time $O(|Q| * |Q'|)$. □

We observe that the asymptotic complexity of our rewriting method is comparable to that of [12] but it handles a larger class of queries (not only downward ones) and works independently of the DTDs.

3.2. MSO

In this section, queries are defined by query automata defined in section 2 as visibly pushdown automata over $T_{\Sigma\times\{\mathbb{0},\mathbb{1}\}}$. The class $\mathcal{QA}$ is used both for annotation and for queries.

Theorem 2. *$\mathcal{QA}$ is closed under query rewriting, i.e. for every root preserving Q_v in $\mathcal{QA}$, for every query Q in $\mathcal{QA}$, there exists a query automaton $\mathsf{Rewrite}(Q, Q_v)$ such that $Q\,(\mathcal{V}iew\,(Q_v, t)) = \mathsf{Rewrite}(Q, Q_v)\,(t)$.*

The automaton $\mathsf{Rewrite}(Q, Q_v)$ is obtained by synchronization of Q and Q_v.

Proof. From the two automata $Q_v = (\Sigma \times \{\mathbb{0}, \mathbb{1}\}, S_v, \Gamma_v, I_v, F_v, R_v)$ and $Q = (\Sigma\times\{\mathbb{0}, \mathbb{1}\}, S, \Gamma, I, F, R)$, we build automaton $Q_\circ = (\Sigma\times\{\mathbb{0}, \mathbb{1}\}, S_\circ, \Gamma_\circ, I_\circ, F_\circ, R_\circ) = \mathsf{Rewrite}(Q, Q_v)$ as follows:

- $S_\circ = S_v \times S$
- $\Gamma_\circ = \Gamma_v \times (\Gamma \cup \{\#\})$
- $I_\circ = I_v \times I$
- $F_\circ = F_v \times F$

- – For every $\eta \in \{\mathrm{op}, \mathrm{cl}\}$, $s_v, s'_v \in S_v$, $a \in \Sigma$, $\gamma_v \in \Gamma_v$,
  for every transition $s_v \xrightarrow{(\eta,(a,\mathbb{0})):\gamma_v} s'_v \in R_v$, for every $s \in S$, we add transition $(s_v, s) \xrightarrow{(\eta,(a,\mathbb{0})):(\gamma_v,\#)} (s'_v, s)$ to $R_\circ$,
  – For every $\eta \in \{\mathrm{op}, \mathrm{cl}\}$, $s_v, s'_v \in S_v$, $a \in \Sigma$, $\gamma_v \in \Gamma_v$, $s, s' \in S, \gamma \in \Gamma, \Diamond \in \{\mathbb{0}, \mathbb{1}\}$,
  for every transition $s_v \xrightarrow{(\eta,(a,\mathbb{1})):\gamma_v} s'_v \in R_v$ and $s \xrightarrow{(\eta,(a,\Diamond)):\gamma} s' \in R$ we add transition $(s_v, s) \xrightarrow{(\eta,(a,\Diamond)):(\gamma_v,\gamma)} (s'_v, s')$ to $R_\circ$,

This automaton $Q_\circ$ satisfies $Q_\circ(t) = Q(\mathcal{V}iew(Q_v, t))$ for every tree t since it satisfies the following invariant.

Invariant: *For every word w over $\{\mathrm{op}, \mathrm{cl}\} \times \Sigma \times \{\mathbb{0}, \mathbb{1}\}$ and every state $(s_v, s) \in S_\circ$, there exists some word u over $\Gamma_\circ$ such that $\mathcal{A}_\circ$ reaches $((s_v, s), u)$ after reading w if and only if there exist a word w' over $\{\mathrm{op}, \mathrm{cl}\} \times (\Sigma \times \Sigma \times \{\mathbb{0}, \mathbb{1}\} \cup \Sigma \times \{\mathbb{0}\}^2)$ and two words u_1 and u_2 over Γ_v and Γ such that the following three conditions are satisfied:*

1. $\pi_{1,3}(w') = w$
2. *Q_v reaches (s_v, u_1) after reading $\pi_{1,2}(w')$, and*
3. *Q reaches (s, u_2) after reading $\pi_{2,3}(w')$.*

□

We have defined a framework for non-materialized security views, where the user's queries are rewritten before being evaluated. This framework thus avoids to materialize one view per role, which improves efficiency when there are numerous roles or when the document or policy are updated frequently. We would like to provide the administrator with a few tools to check that the SAS he defines really match her expectations. In particular, we provide the administrator with techniques for comparing access policies, something that may be useful for instance to establish whether a modification of the policy allows to disclose more information than was previously available.

4. Static analysis of security access specifications: the general case

We wish to provide the administrator with tools for comparing access control policies. A straightforward approach is to compare the nodes made visible by the root preserving queries:

Definition 2. Given two root preserving queries Q_1 and Q_2 with $\mathrm{dom}(Q_1) = \mathrm{dom}(Q_2) = D$, we say that $Q_1 \preccurlyeq_1 Q_2$ if

$$\forall t \in D.\, Q_1(t) \subseteq Q_2(t)$$

which means that all nodes visible for Q_1 are also displayed by query Q_2.

Example 3. We consider the DTD D_0 given in example 2, with another access function X_1. In this access function, nodes `src` under `dev` are always hidden (not only where they are under a proprietary licensed project). So the last rule of X_0 is replaced by :

$\texttt{dev} \rightarrow \texttt{src}, \texttt{doc}$
$X_1(\texttt{dev}, \texttt{src})$=false
$X_1(\texttt{dev}, \texttt{doc})$=true

In this example, access function X_1 hides more nodes than X_0, so $Q_{(D_0,X_1)} \preccurlyeq_1 Q_{(D_0,X_0)}$ (compare figure 3 and figure 4). But hiding nodes may reveal some information. Indeed, for every t valid for the DTD D_0, the projects with free license that are currently under development can be selected with the following $\mathcal{X}Reg$ expression on $\mathcal{V}iew\,(Q_{X_1}, t)$:

$$\Downarrow\!::\texttt{projects}/\Downarrow\!::\texttt{project}[\texttt{not}(\Downarrow\!::\texttt{src})\ \texttt{and}\ \Downarrow\!::\texttt{license}/\Downarrow\!::\texttt{free}]$$

So the user can distinguish some projects under development from stable projects, which was not possible with X_0.

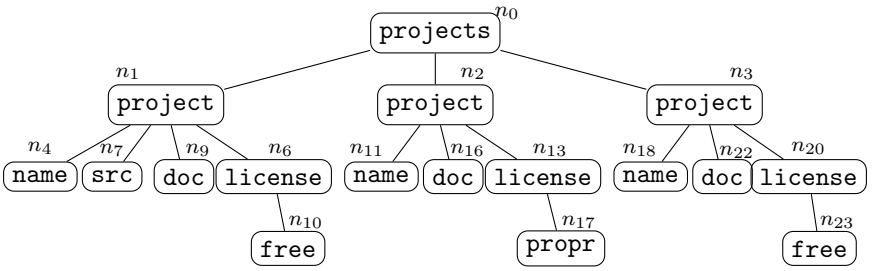


Figure 4: The view $\mathcal{V}iew\,(Q_{X_1}, t_0)$.

For this reason, we define now another way to compare root preserving queries. Given a class of queries $\mathcal{C}$ and root preserving query Q in $\mathcal{C}$, we define the class of 'expressible queries' over the source document as

$$Public^{\mathcal{C}}(Q) = \{Q' \in \mathcal{C} \mid \exists Q'' \in \mathcal{C}.\, \forall t \in \mathrm{dom}(Q).\, Q''(\mathcal{V}iew\,(Q,t)) = Q'(t)\}$$

We fix a class of queries $\mathcal{C}$ and assume that

1. query Q_{all} belongs to $\mathcal{C}$ where for every tree t, $Q_{all}(t)$ selects all the nodes of t

2. $\mathcal{C}$ is closed under query rewriting.

Definition 3. Given two root preserving queries Q_1 and Q_2 in $\mathcal{C}$ with $\mathrm{dom}(Q_1) = \mathrm{dom}(Q_2)$, we say that $Q_1 \preccurlyeq_{2,\mathcal{C}} Q_2$ if

$$Public^{\mathcal{C}}(Q_1) \subseteq Public^{\mathcal{C}}(Q_2)$$

This definition requires further explanation. On the whole, it means that all information we could retrieve from Q_1 using some query from class $\mathcal{C}$ could also be retrieved from Q_2 using some query from class $\mathcal{C}$. Thus Q_1 does not disclose information hidden by Q_2. The following characterization gives a useful alternative to this definition: $\preccurlyeq_{2,\mathcal{C}}$ can be expressed in terms of query rewriting.

Proposition 1. *Given two root preserving queries Q_1 and Q_2 with $\mathrm{dom}(Q_1) = \mathrm{dom}(Q_2) = D$, $Q_1 \preccurlyeq_{2,\mathcal{C}} Q_2$ if and only if $Q_1 \in Public^{\mathcal{C}}(Q_2)$, i.e. $\exists Q \in \mathcal{C}. \forall t \in D. Q(\mathcal{V}iew\,(Q_2, t)) = Q_1(t)$. This means that $Q_1 \preccurlyeq_{2,\mathcal{C}} Q_2$ if and only if a user with view induced by Q_2 can simulate view induced by Q_1.*

Proof. Suppose $Q_1 \preccurlyeq_{2,\mathcal{C}} Q_2$. Since we assumed Q_{all} belongs to $\mathcal{C}$, $Q_1 \in Public^{\mathcal{C}}(Q_1)$. So $Q_1 \in Public^{\mathcal{C}}(Q_2)$. Conversely suppose $Q_1 \in Public^{\mathcal{C}}(Q_2)$, and let Q denote some query in $\mathcal{C}$ such that for all t in D, $Q(\mathcal{V}iew\,(Q_2, t)) = Q_1(t)$. Observe that, since Q_1 is root preserving, so is Q. Fix also $Q' \in Public^{\mathcal{C}}(Q_1)$ and let Q'' denote some query such that for all t in D, $Q''(\mathcal{V}iew\,(Q_1, t)) = Q'(t)$. Then, since we supposed $\mathcal{C}$ to be closed under query rewriting, there exists a query Q_r in $\mathcal{C}$ such that for all t, $Q_r(t) = Q''(\mathcal{V}iew\,(Q, t))$. Therefore, for all t in D, $Q_r(\mathcal{V}iew\,(Q_2, t)) = Q'(t)$, hence $Q' \in Public^{\mathcal{C}}(Q_2)$, which concludes our proof. □

When we consider the general case ($\mathcal{C}$ is the set of all the queries closed under isomorphism), we denote the comparison by $\preccurlyeq_2$. In this case, the following proposition makes the link with the notion of *determinacy* [25] : Q_2 determines Q_1 when $\forall t, t' \in D, \mathcal{V}iew\,(Q_2, t) = \mathcal{V}iew\,(Q_2, t') \implies \mathcal{V}iew\,(Q_1, t) = \mathcal{V}iew\,(Q_1, t')$.

We shall use the following lemma:

Lemma 5. *Let Q_1 and Q_2 be two root preserving queries with $\mathrm{dom}(Q_1) = \mathrm{dom}(Q_2) = D$. Then $Q_1 \preccurlyeq_2 Q_2$ implies $Q_1 \preccurlyeq_1 Q_2$.*

Proof. Let Q_1 and Q_2 be two queries with domain D such that $Q_1 \preccurlyeq_2 Q_2$. Suppose that $Q_1 \not\preccurlyeq_1 Q_2$, then there exists a tree t and a node n in $Q_1(t)$ that is not in $Q_2(t)$. Let t' be a tree obtained from t by replacing n with a "fresh" node $n' \notin N_t$ having the same label as n. As Q_2 and Q_1 are closed by isomorphism, $\mathcal{V}iew\,(Q_2, t') = \mathcal{V}iew\,(Q_2, t)$ and $Q_1(t') \neq Q_1(t)$, in contradiction with our hypothesis. Therefore, we have $Q_1 \preccurlyeq_1 Q_2$. □

Proposition 2. *Given two root preserving queries Q_1 and Q_2 in $\mathcal{X}Reg$ or MSO such that $\mathrm{dom}(Q_1) = \mathrm{dom}(Q_2) = D$, $Q_1 \preccurlyeq_2 Q_2$ if and only if for all $t, t' \in D, \mathcal{V}iew\,(Q_2, t) = \mathcal{V}iew\,(Q_2, t')$ implies $\mathcal{V}iew\,(Q_1, t) = \mathcal{V}iew\,(Q_1, t')$.*

Proof. Assume first that $Public^{\mathcal{C}}(Q_1) \subseteq Public^{\mathcal{C}}(Q_2)$ for some $\mathcal{C}$ that contains Q_1. In particular, $Q_1 \in Public^{\mathcal{C}}(Q_1)$, hence $Q_1 \in Public^{\mathcal{C}}(Q_2)$, and therefore for all $t, t' \in D$, $\mathcal{V}iew\,(Q_2, t) = \mathcal{V}iew\,(Q_2, t')$ implies $\mathcal{V}iew\,(Q_1, t) = \mathcal{V}iew\,(Q_1, t')$.

Conversely, let $\mathcal{C}$ be the class of all queries, Q' be any query in $Public^{\mathcal{C}}(Q_1)$ and Q'' a query such that for all $t \in D$, $Q''(\mathcal{V}iew\,(Q_1, t)) = Q'(t)$. From hypothesis, for each tree $t_2 \in \mathcal{V}iew\,(Q_2, D)$, there exists a tree t_1 such that for all tree

t with $\mathcal{V}iew\,(Q_2,t) = t_2$ then $\mathcal{V}iew\,(Q_1,t) = t_1$. Therefore, for all tree t with $\mathcal{V}iew\,(Q_2,t) = t_2$, the value of $Q'(t)$ is the same. Moreover, from Lemma 5, $Q_1 \preccurlyeq_1 Q_2$ and it follows $Q'(t) \subseteq Q_1(t) \subseteq Q_2(t) = N_{t_2}$. Thus, $Q'(t)$ is a subset of $Q_2(t)$ and only depends on $\mathcal{V}iew\,(Q_2,t)$. So, we can define a query Q_r that "computes" $Q'(t)$ from $\mathcal{V}iew\,(Q_2,t)$ (the nodes selected by Q_r on tree t_2 are obtained by choosing arbitrarily a tree t such that $\mathcal{V}iew\,(Q_2,t) = t_2$ and then selecting the nodes in $Q'(t)$). This query Q_r is closed under isomorphism since Q' and Q_2 are so. Thus $Q' \in Public^{\mathcal{C}}(Q_1)$ that implies $Public^{\mathcal{C}}(Q_1) \subseteq Public^{\mathcal{C}}(Q_2)$ that is $Q_1 \preccurlyeq_2 Q_2$. □

Clearly, $Q_1 \preccurlyeq_{2,\mathcal{X}Reg} Q_2 \implies Q_1 \preccurlyeq_{2,MSO} Q_2 \implies Q_1 \preccurlyeq_2 Q_2$. These strong comparisons may be relaxed whenever the node identifier does not really play a role. Furthermore, a natural option would be to take all data, be it visible or invisible, into account when we compare views: the knowledge of the access control policy may allow to deduce some information on the hidden parts of the document from the structure of the view document. We therefore propose a third comparison for policies, based on *certain answers*[20]

Definition 4. Given a root preserving query Q_v, a boolean query $\mathcal{Q}$, and a tree t_v in $\mathcal{V}iew\,(Q_v, \mathrm{dom}(Q_v))$, we define the set of *source documents* of t_v for Q_v as $\mathrm{Src}\,(t_v,Q_v) = \{t \in \mathrm{dom}(Q_v) \mid \mathcal{V}iew\,(Q_v,t) \sim t_v\}$. The *certain answer* of query $\mathcal{Q}$ for t_v is

$$Certain_{Q_v}\,(\mathcal{Q};t_v) = \bigwedge_{t \in \mathrm{Src}(t_v,Q_v)} \mathcal{Q}(t).$$

We can now introduce our comparison, stating that root preserving Q_1 is more restrictive than root preserving Q_2 if for every source document t the certain answers for t with the first view are also certain answers for the second one.

Definition 5. Given a class of queries $\mathcal{C}$ and two root preserving Q_1 and Q_2 with $\mathrm{dom}(Q_1) = \mathrm{dom}(Q_2) = D$, we say that $Q_1 \preccurlyeq_{3,\mathcal{C}} Q_2$ if

$$\forall t \in D.\, \forall \mathcal{Q} \in \mathcal{C}.\, Certain_{Q_1}\,(\mathcal{Q};\mathcal{V}iew\,(Q_1,t)) \implies Certain_{Q_2}\,(\mathcal{Q};\mathcal{V}iew\,(Q_2,t))$$

We define the notion of view inversion in order to prove that in our setting comparison $\preccurlyeq_{3,\mathcal{C}}$ does not depend on the class of queries $\mathcal{C}$ considered.

Definition 6. Given two classes of queries $\mathcal{C}$ and $\mathcal{C}'$, we say that $\mathcal{C}$ *permits* $\mathcal{C}'$*-view inversion* if for every root preserving query $Q_1 \in \mathcal{C}'$, any tree $t' \in \mathcal{V}iew\,(Q_1, \mathrm{dom}(Q_1))$, there is a boolean query $\mathrm{Ant}\,(t',Q_1)$ in $\mathcal{C}$ such that $\forall t \in \mathrm{dom}(Q_1).\, \mathrm{Ant}\,(t',Q_1)\,(t) = \mathsf{true}$ iff $t \in \mathrm{Src}\,(t',Q_1)$, i.e., iff $\mathcal{V}iew\,(Q_1,t) \sim t'$.

This means query $\mathrm{Ant}\,(t',Q_1)$ is satisfied on trees whose view (for Q_1) is isomorphic to t'.

Lemma 6. *Every class $\mathcal{C} \in \{\mathcal{X}Reg, MSO\}$ permits $\mathcal{C}$-view inversion.*

PROOF. Let Q_1 denote a root-preserving $\mathcal{X}Reg$ query and let t' denote a tree in $\mathcal{V}iew\,(Q_1, \mathrm{dom}(Q_1))$. We can easily define a boolean query $f \in \mathcal{X}Reg$ such that for every tree t, $f \models t$ if and only if $t \sim t'$. The construction for the composition of queries in section 3 can be applied to boolean queries as well as root-preserving queries; thus, by rewriting the base axes of f, we obtain a $\mathcal{X}Reg$ query $\mathsf{Rewrite}(f, Q_1)$ which for every tree t satisfies $\mathsf{Rewrite}(f, Q_1)(t) = \mathsf{true}$ if and only if $\mathcal{V}iew\,(Q_1, t) \sim t'$. The proof for *MSO* follows the same lines. □

The definition for $\preccurlyeq_{3,\mathcal{C}}$ is not very practical, due to the quantification over queries. Therefore we introduce the following characterizations.

Proposition 3. *For all classes $\mathcal{C}, \mathcal{C}'$, if $\mathcal{C}$ permits $\mathcal{C}'$-view inversion, then for every root preserving queries $Q_1, Q_2 \in \mathcal{C}'$ with $\mathrm{dom}(Q_1) = \mathrm{dom}(Q_2) = D$:*

$$\begin{aligned} Q_1 \preccurlyeq_{3,\mathcal{C}} Q_2 \quad &\iff \quad \forall t \in D.\, \mathit{Certain}_{Q_2}\left(\mathrm{Ant}\left(\mathcal{V}iew\,(Q_1, t), Q_1\right); \mathcal{V}iew\,(Q_2, t)\right). \\ &\iff \quad \forall t, t' \in D.\, \mathcal{V}iew\,(Q_2, t) \sim \mathcal{V}iew\left(Q_2, t'\right) \ \textit{implies} \\ &\qquad\quad \mathcal{V}iew\,(Q_1, t) \sim \mathcal{V}iew\left(Q_1, t'\right) \end{aligned}$$

PROOF. Assume $Q_1 \preccurlyeq_{3,\mathcal{C}} Q_2$. Since $\mathcal{C}$ permits $\mathcal{C}'$-view inversion, for all t in D, $\mathrm{Ant}\,(t, Q_1)$ exists and $\mathit{Certain}_{Q_1}\left(\mathrm{Ant}\left(\mathcal{V}iew\,(Q_1, t), Q_1\right); \mathcal{V}iew\,(Q_1, t)\right)$ holds. Hence, $\mathit{Certain}_{Q_2}\left(\mathrm{Ant}\left(\mathcal{V}iew\,(Q_1, t), Q_1\right); \mathcal{V}iew\,(Q_2, t)\right)$ also holds.

Suppose now that $\mathit{Certain}_{Q_2}\left(\mathrm{Ant}\left(\mathcal{V}iew\,(Q_1, t), Q_1\right); \mathcal{V}iew\,(Q_2, t)\right)$ for all t in D, and fix some t, t' in D such that $\mathcal{V}iew\,(Q_2, t) \sim \mathcal{V}iew\,(Q_2, t')$. Then, $\mathrm{Ant}\left(\mathcal{V}iew\,(Q_1, t), Q_1\right)(t') = \mathsf{true}$, hence $\mathcal{V}iew\,(Q_1, t) \sim \mathcal{V}iew\,(Q_1, t')$.

To conclude, suppose that $\mathcal{V}iew\,(Q_2, t) \sim \mathcal{V}iew\,(Q_2, t') \implies \mathcal{V}iew\,(Q_1, t) \sim \mathcal{V}iew\,(Q_1, t')$ for all t, t' in D. Then, for every $t \in D$, $\mathrm{Src}\left(\mathcal{V}iew\,(Q_2, t), Q_2\right) \subseteq \mathrm{Src}\left(\mathcal{V}iew\,(Q_1, t), Q_1\right)$. Consequently, every t in D and Q in $\mathcal{C}$ satisfy the property $\mathit{Certain}_{Q_1}\left(\mathcal{Q}; \mathcal{V}iew\,(Q_1, t)\right) \implies \mathit{Certain}_{Q_2}\left(\mathcal{Q}; \mathcal{V}iew\,(Q_2, t)\right)$. □

Remark: note that under those assumptions on the classes of queries, $\preccurlyeq_{3,\mathcal{C}}$ does not depend upon $\mathcal{C}$. Actually, Proposition 3 characterizes comparisons in terms of determinacy (defined modulo isomorphism). Henceforth, $\preccurlyeq_3$ will denote $\preccurlyeq_{3,\mathcal{C}}$ for all class $\mathcal{C}$ that permits $\mathcal{C}$-view inversion.
The following results describe how the three definitions for policy comparison are related:

Proposition 4. *Given any class of queries $\mathcal{C}$ and root preserving queries Q_1 and Q_2 in $\mathcal{C}$ with $\mathrm{dom}(Q_1) = \mathrm{dom}(Q_2)$,*

1. $Q_1 \preccurlyeq_2 Q_2 \implies Q_1 \preccurlyeq_1 Q_2$
2. $Q_1 \preccurlyeq_2 Q_2 \implies Q_1 \preccurlyeq_3 Q_2$
3. $(Q_1 \preccurlyeq_1 Q_2 \wedge Q_1 \preccurlyeq_3 Q_2) \not\Rightarrow Q_1 \preccurlyeq_2 Q_2$
4. $Q_1 \preccurlyeq_2 Q_2 \not\Rightarrow Q_1 \preccurlyeq_{2,MSO} Q_2$.

PROOF. 1. This is Lemma 5.

2. Let $Q_1 \preccurlyeq_2 Q_2$. Let $\mathcal{Q}$ be a Boolean query and t in $\mathrm{dom}(Q_1)$ such that $\mathit{Certain}_{Q_1}(\mathcal{Q}; \mathcal{V}\mathit{iew}(Q_1, t))$. Let t_0 be a tree such that $\mathcal{V}\mathit{iew}(Q_2, t) \sim \mathcal{V}\mathit{iew}(Q_2, t_0)$. There exists a tree t' with $t' \sim t_0$ and $\mathcal{V}\mathit{iew}(Q_2, t') = \mathcal{V}\mathit{iew}(Q_2, t)$, because we considered queries closed under isomorphism. From Proposition 2, it follows $\mathcal{V}\mathit{iew}(Q_1, t) = \mathcal{V}\mathit{iew}(Q_1, t')$ and, since $t' \sim t_0$, $\mathcal{V}\mathit{iew}(Q_1, t) \sim \mathcal{V}\mathit{iew}(Q_1, t_0)$. We have proved $Q_1 \preccurlyeq_3 Q_2$.
3. Let D be the DTD defined by $\mathtt{r} \rightarrow \mathtt{a*,b,a,a*}$, let $\chi_1 = \Downarrow[\mathsf{self}::a \ \mathsf{and} \ \Leftarrow::b]$ and $\chi_2 = \Downarrow::a$. Let Q_1 be the query that synthesizes validation against D and $XReg$ expression χ_1 and Q_2 be the query that synthesizes validation against D and $XReg$ expression χ_2. Those queries satisfy: $(Q_1 \preccurlyeq_1 Q_2 \wedge Q_1 \preccurlyeq_3 Q_2)$ but $Q_1 \not\preccurlyeq_2 Q_2$.
4. We show a stronger result actually; we prove that determinacy for "simple" annotations does not imply the existence of an *MSO* query rewriting even when $\mathcal{V}\mathit{iew}(Q_2, D)$ is regular.
   Let D be the DTD defined by $\mathtt{r} \rightarrow \mathtt{a,r,a} \mid \mathsf{empty}$, let X be the access function defined by $X(r, r) = \mathsf{false}$, and $X(r, a) = \mathsf{true}$, let $\chi_1 = \Downarrow^*[\mathsf{self}::a \wedge \neg \Leftarrow]$. Let Q_1 be the query that synthesizes validation against D and $XReg$ expression χ_1 and $Q_2 = Q_{(D,X)}$. $\mathcal{V}\mathit{iew}(Q_2, L(D))$ consists of all trees of depth 1 with nodes labeled a below root r, in even number. Any query Q such that $\mathsf{Rewrite}(Q, Q_2) = Q_1$ would have to select the n first 'a' elements in a^{2n}, which is beyond the power of regular queries. □

We define the following decision problems, parameterized by $i \in \{1, 2, 3\}$ and a class of queries $\mathcal{C}$:

Problem: $\preccurlyeq_{i,\mathcal{C}}$

Input: Root preserving queries $Q_1, Q_2 \in \mathcal{C}$ with $\mathrm{dom}(Q_1) = \mathrm{dom}(Q_2)$.
Question: $Q_1 \preccurlyeq_i Q_2$ *?*

Proposition 5. *For any class of queries $\mathcal{C} \in \{\mathcal{X}Reg, MSO\}$ there is a polynomial time reduction from $\preccurlyeq_{1,\mathcal{C}}$ to $\preccurlyeq_{2,\mathcal{C}}$, and a polynomial time reduction from $\preccurlyeq_{1,\mathcal{C}}$ to $\preccurlyeq_{3,\mathcal{C}}$.*

Proof. Let Q_1 and Q_2 denote two root preserving queries with identical domain D. We denote by Σ' the new alphabet: $\Sigma' = ((\Sigma \backslash \{r\}) \times \{1, 2\}) \cup \{\$\} \cup \{(r, 1)\}$ where r is the label of the root of trees in D. Intuitively, the \$ will be used as a tag that marks the positions selected by Q_1, while the substitution with two copies of each letter will be necessary only for the reduction to $\preccurlyeq_{3,\mathcal{C}}$.

We define a transformation τ that adds a \$ symbol as the leftmost child of every node of the trees in D: $\forall a \in \Sigma, \tau(a(t_1, t_2, \ldots, t_n)) = a(\$, \tau(t_1), \ldots, \tau(t_n))$. We also define morphism ϕ from Σ' to $\Sigma \cup \{\$\}$ that projects the labels on their first component. Formally, $\phi(\$) = \$$, $\phi((r, 1)) = r$, and for all a in $\Sigma \backslash \{r\}, \phi((a, 1)) = \phi((a, 2)) = a$. Finally, D' is defined as $\phi^{-1}(\tau(D))$.

Given any tree $t \in D'$, $\tau^{-1}(t)$ returns the tree obtained from t by removing the \$ nodes (only leaves may be labeled by a \$), and $\phi(\tau^{-1}(t))$ additionally projects the labels on the first component. We define two queries Q'_1 and Q'_2 as

follows. For every $i \in \{1,2\}$, $Q'_i(t) \cap N_{\tau^{-1}(t)} = Q_i(\phi(\tau^{-1}(t)))$. Q'_1 selects no node with label \$, and Q'_2 selects a node with label \$ if and only if its parent node is selected by Q'_1. If $\mathcal{C}$ is one of $\mathcal{X}Reg$ or MSO (query automata), then queries Q'_1 and Q'_2 in $\mathcal{C}$ can clearly be defined in polynomial time from Q_1 and Q_2. To conclude the proof we observe that: $Q_1 \preccurlyeq_1 Q_2 \iff Q'_1 \preccurlyeq_2 Q'_2 \iff Q'_1 \preccurlyeq_3 Q'_2$.

Here is a proof for the observation: if $Q_1 \preccurlyeq_1 Q_2$ does not hold, then there exists a tree t' and node $n \in N_t$ such that $n \in Q_1(t') \backslash Q_2(t')$. Let t_1 be a tree such that $\phi(\tau^{-1}(t_1)) = t'$ and $\lambda_{t_1}(n) = (a,1)$, and t_2 be obtained from t_1 by relabeling n with $(a,2)$. From $Q'_2(t_1)$ one cannot guess if the label of n is $(a,1)$ or $(a,2)$: $\mathcal{V}iew\,(Q'_2,t_1) \sim \mathcal{V}iew\,(Q'_2,t_2)$, and yet $\mathcal{V}iew\,(Q'_1,t_1) \not\sim \mathcal{V}iew\,(Q'_1,t_2)$. Therefore, $Q'_1 \preccurlyeq_3 Q'_2$ implies $Q_1 \preccurlyeq_1 Q_2$. When $Q_1 \preccurlyeq_1 Q_2$, $Q'_1 \preccurlyeq_{2,\mathcal{C}} Q'_2$ obviously holds, since in that case we only need to select in the view for Q'_2 the nodes having a child labeled \$ to get the nodes selected by Q'_1. Moreover, $Q'_1 \preccurlyeq_{2,\mathcal{C}} Q'_2$ implies $Q'_1 \preccurlyeq_3 Q'_2$ by Proposition 4. We observe that we have used $\preccurlyeq_{2,\mathcal{C}}$ instead of $\preccurlyeq_2$ in the previous paragraph, which yields the additional result that $Q_1 \preccurlyeq_1 Q_2 \iff Q'_1 \preccurlyeq_{2,\mathcal{C}} Q'_2$. Consequently we also have a reduction from $\preccurlyeq_{1,\mathcal{C}}$ to the problem of deciding comparison $\preccurlyeq_{2,\mathcal{C}}$. □

Example 4. Figure 5 illustrates the reduction for two $\mathcal{X}Reg$ queries. Clearly, the queries Q_1 and Q_2 from that figure satisfy $Q_1 \preccurlyeq_1 Q_2$. Therefore, queries Q'_1 and Q'_2 satisfy $Q'_1 \preccurlyeq_3 Q'_2$ and even $Q'_1 \preccurlyeq_{2,\mathcal{X}Reg} Q'_2$. Query $\Downarrow^*::[\Downarrow::\$]$ is a rewriting of Q'_1 in terms of Q'_2.

When the class of queries $\mathcal{C}$ is expressive enough, for instance $\mathcal{C} \in \{\mathcal{X}Reg, MSO\}$, we can reduce determinacy to the third comparison:

Proposition 6. *Given two root preserving queries Q_1, Q_2 in $\mathcal{C}$ (where $\mathcal{C}$ in $\{\mathcal{X}Reg, MSO\}$) with $\mathrm{dom}(Q_1) = \mathrm{dom}(Q_2)$, we can compute in polynomial time two queries Q'_1 and Q'_2 in $\mathcal{C}$ such that $Q_1 \preccurlyeq_2 Q_2 \iff (Q_1 \preccurlyeq_1 Q_2 \wedge Q'_1 \preccurlyeq_3 Q'_2)$.*

Proof. Fix $\mathcal{C} \in \{\mathcal{X}Reg, MSO\}$, and root preserving queries Q_1, Q_2 such that $\mathrm{dom}(Q_1) = \mathrm{dom}(Q_2) = D$. We first test the inclusion, and then we must check not only isomorphism constraints, but also that "the same nodes appear at the same position". For this purpose we modify D, inserting dummy nodes into the first view so as to indicate the positions before we test $\preccurlyeq_{3,\mathcal{C}}$.

Formally, the proof works as follows: let \$ represent a new symbol outside Σ. We define D' from D such that for all tree t in D, every subtree $a(t_1,\ldots,t_n)$ rooted at even depth is replaced by $\$(a(t_1,\ldots,t_n))$. Note that if D is expressible in $\mathcal{C}$, D' is expressible in $\mathcal{C}$. Next, we define from Q_1 and Q_2 queries Q'_1 and Q'_2 of domain D' as follows: given a tree t' in D', let t be the tree obtained from t' by deleting every \$-labeled node (so each node of odd depth in t' gets adopted by its grandfather node). Q'_2 is defined by $Q'_2(t') = Q_2(t)$, i.e. Q'_2 hides all \$-labeled nodes and the nodes hidden by Q_2 in t, and $Q'_1(t')$ contains exactly $Q_1(t)$ plus every node n with label \$ such that there exists in t' a node n' below n satisfying $n' \in Q_2(t)$. So, we have constructed two queries Q'_1 and Q'_2 such that $Q_1 \preccurlyeq_2 Q_2 \iff (Q_1 \preccurlyeq_1 Q_2 \wedge Q'_1 \preccurlyeq_3 Q'_2)$.

$Q_2 = \Downarrow^* :: a / \Downarrow^* :: b \qquad Q'_2 = \Downarrow^* :: [\mathsf{self} :: (a,1) \text{ or } \mathsf{self} :: (a,2)] / \Downarrow^* :: [\mathsf{self} :: (b,1) \text{ or } \mathsf{self} :: (b,2)] \cup Q'_1 / \Downarrow :: \$$

$Q_1 = \Downarrow^* :: a / \Downarrow :: b \qquad Q'_1 = \Downarrow^* :: [\mathsf{self} :: (a,1) \text{ or } \mathsf{self} :: (a,2)] / \Downarrow :: [\mathsf{self} :: (b,1) \text{ or } \mathsf{self} :: (b,2)]$

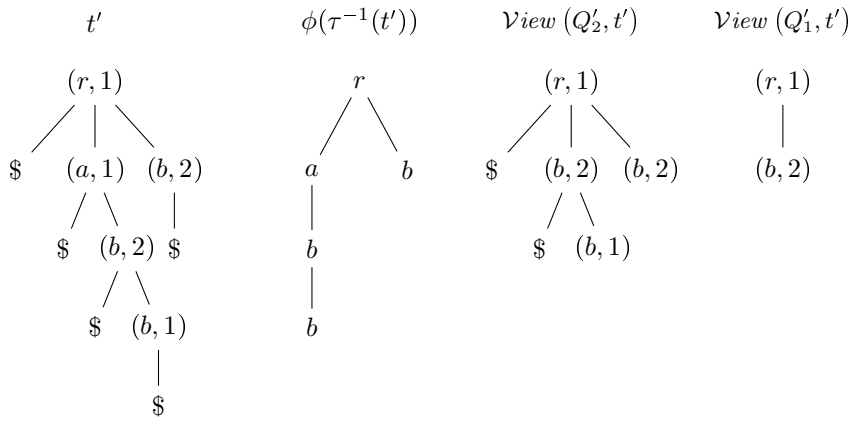


Figure 5: Reduction from $\preccurlyeq_1$ to $\preccurlyeq_{2,\mathcal{X}Reg}$ and $\preccurlyeq_3$ for particular Q_1 and Q_2.

At first glance, this looks like a Turing reduction, because we use two instances of $\preccurlyeq_3$: one for $Q'_1 \preccurlyeq_3 Q'_2$ and one for $Q_1 \preccurlyeq_1 Q_2$ (we recall from Proposition 5 that comparison $\preccurlyeq_1$ reduces into $\preccurlyeq_3$). However, it is easy to build a single instance from these two: we can use disjoint alphabets for the two instances by copying the alphabet, and then use as domain the set of trees whose root has two children; each child being devoted to one instance. □

Theorem 3. *Given $\mathcal{C} \in \{\mathcal{X}Reg, MSO\}$, and two root preserving queries Q_1 and Q_2 in $\mathcal{C}$, testing $Q_1 \preccurlyeq_{2,\mathcal{C}} Q_2$ is undecidable.*

Proof. We use a reduction from regular separability of two context-free grammars. Recall that two context-free grammars G_1 and G_2 over the alphabet Γ are *regularly separable* if there exists a regular language R (over Γ) such that $L(G_1) \subseteq R$ and $L(G_2) \subseteq R^{\complement}$, where $R^{\complement}$ is the complement of R. Checking regular separability of two context-free languages is known to be undecidable [30].

We give the proof for $\mathcal{C} = \mathcal{X}Reg$; the result for *MSO* follows the same lines. The reduction constructs a DTD D defining the set of all derivation trees of G_1 and G_2. The query Q_2 hides all nonterminals from the derivation tree except the root, thus yielding a tree of depth one whose leaves form a word of $L(G_1) \cup L(G_2)$. The query Q_1 works similarly except that it also hides terminals derived from nonterminals of G_2; essentially, it yields only words of $L(G_1)$.

If G_1 and G_2 are separable by a regular set R, then the regular expression describing R can be easily rewritten into a $\mathcal{X}Reg$ query Q such that for all t in D, $Q(\mathcal{V}iew\,(Q_2, t)) = Q_1(t)$, that is $Q_{(D,Q_1)} \preccurlyeq_{2,\mathcal{C}} Q_{(D,Q_2)}$. Conversely, suppose there is a $\mathcal{X}Reg$ query Q such that for all t in D, $Q(\mathcal{V}iew\,(Q_2, t)) = Q_1(t)$. Essentially, Q selects words from $L(G_1)$ and hides words from $L(G_2)$, hence it separates G_1 and G_2. Then Q is equivalent to a tree *MSO* formula φ [5], and we remark that φ is interpreted on trees of height 1 only. Therefore, there exists a word *MSO* formula ψ that captures exactly the words consisting of labels of the consecutive children of the root node. This formula ψ can be converted into a regular expression [33] which defines a set separating G_1 and G_2. □

We prove similarly that determinacy is undecidable:

Theorem 4. *Given root preserving $\mathcal{X}Reg$ queries Q_1 and Q_2, testing $Q_1 \preccurlyeq_2 Q_2$ is undecidable.*

Proof. The proof is similar to the one for Theorem 3, hiding derivations of context-free grammars, except that the reduction is toward emptiness of intersection: recall that the problem that takes as input two context-free grammars G_1 and G_2 and answers whether $L(G_1) \cap L(G_2) = \varnothing$ is undecidable.

The reduction constructs a DTD D defining the set of all derivation trees of G_1 and G_2. The query Q_2 hides all nonterminals from the derivation tree except the root, thus yielding a tree of depth one whose leaves form a word of $L(G_1) \cup L(G_2)$. The query Q_1 works similarly except that it also hides terminals derived from nonterminals of G_2; essentially, it yields only words of $L(G_1)$. If G_1 and G_2 are disjoint then for any two trees t, t' in D such that

$\mathcal{V}iew\,(Q_2, ()\,t) = \mathcal{V}iew\,(Q_2, t')$, either t and t' both correspond to derivation trees of G_1 or they both correspond to derivation trees of G_2. Either way, $\mathcal{V}iew\,(Q_1, t) = \mathcal{V}iew\,(Q_1, t')$. Conversely, suppose there exists $w \in L(G_1) \cap L(G_2)$. Then there exist a derivation tree t (resp. t') of w for G_1 (resp. for G_2). Consequently, $Q_1 \not\leqslant_2 Q_2$ since $\mathcal{V}iew\,(Q_2, ()\,t) = \mathcal{V}iew\,(Q_2, t')$ and $\mathcal{V}iew\,(Q_1, t) \neq \mathcal{V}iew\,(Q_1, t')$. This concludes the proof. □

Proposition 7. *We denote by $\equiv_3$ the equivalence relation $Q_1 \equiv_3 Q_2 \iff Q_1 \leqslant_3 Q_2 \wedge Q_2 \leqslant_3 Q_1$. In general (and even if the visibility of a node depends only on its label) testing whether $Q_1 \equiv_3 Q_2$ is undecidable, therefore testing whether $Q_1 \leqslant_3 Q_2$ is undecidable.*

Proof. Given an instance of PCP $\mathcal{P}: u_1, \dots u_n, v_1, \dots v_n$ with $u_i, v_i \in \Sigma^*$ for all $i \leqslant n$, we define as follows a DTD D over alphabet $\Sigma \cup \{\mathtt{u}, \mathtt{v}, \#, \mathtt{1}, \dots \mathtt{n}\}$, together with access functions X_1, X_2. The DTD production rules are: $\mathtt{r} \to \mathtt{u} \mid \mathtt{v}$, $\mathtt{u} \to (\mathtt{u_1}, \mathtt{u}, \mathtt{1}) \mid \dots \mid (\mathtt{u_n}, \mathtt{u}, \mathtt{n}) \mid \#$, and $\mathtt{v} \to (\mathtt{v_1}, \mathtt{v}, \mathtt{1}) \mid \dots \mid (\mathtt{v_n}, \mathtt{v}, \mathtt{n}) \mid \#$, and the access functions are, for all j in $\{1, 2\}$ and $\alpha \in \Sigma \cup \{\#\} \cup \{1, \dots n\}$:

$$X_1(r, u) = X_1(r, v) = \mathsf{false},\ X_2(r, u) = X_2(r, v) = \mathsf{true}$$
$$X_j(u, u) = X_j(v, v) = \mathsf{false},\ X_j(u, \alpha) = X_j(v, \alpha) = \mathsf{true}$$

Note that the view for access function X_1 consists of some tree of depth 2 (hence can be identified with words), see Figure 6 for an illustration of the PCP instance ($u_1 = aab, u_2 = ba, u_3 = b$, $v_1 = aa, v_2 = bb, v_3 = abb$) over alphabet $\Sigma = \{a, b\}$: the two annotations derived from this instance do not satisfy $Q_{(D,X_1)} \equiv_3 Q_{(D,X_2)}$. $Q_{(D,X_1)}(t)$ can easily be obtained from $Q_{(D,X_2)}$ by erasing u or v, so $Q_{(D,X_1)} \leqslant_3 Q_{(D,X_2)}$ trivially holds. Clearly, $Q_{(D,X_2)} \leqslant_3 Q_{(D,X_1)}$ if and only if there is no solution to the PCP problem, because the only difference between $Q_{(D,X_1)}$ and $Q_{(D,X_2)}$ is that the latter selects the child of the root, so $Q_{(D,X_2)} \leqslant_3 Q_{(D,X_1)}$ if and only if one can distinguish for every sequence $i_1 \dots i_k$ and every word $w \in S = \{u_{i_1} u_{i_2} \dots u_{i_k}, v_{i_1} v_{i_2} \dots v_{i_k}\}$ if w has been from the u_i or from the v_i. In other words, $Q_{(D,X_2)} \leqslant_3 Q_{(D,X_1)}$ if and only if $u_{i_1} u_{i_2} \dots u_{i_k} \neq v_{i_1} v_{i_2} \dots v_{i_k}$ for every sequence $i_1 \dots i_k$, which is the definition of PCP. Hence, $Q_{(D,X_1)} \equiv_3 Q_{(D,X_2)}$ if and only if the answer of $\mathcal{P}$ is negative. Thus testing $Q_{(D,X_1)} \equiv_3 Q_{(D,X_2)}$ is undecidable. □

5. Restrictions on the views

Defining the view with unrestricted $\mathcal{X}Reg$ queries or automata raises a major difficulty: the sets of view trees $\mathcal{V}iew\,(V, D) = \{\mathcal{V}iew\,(V, t)\,t \in D\}$ need not be regular. Approximating the view schema may be a solution, yet this non-regularity also makes decision problems such as policy comparison intractable, in addition to preventing the construction of the view schema. Therefore, we investigate a few restrictions on the views that guarantee $\mathcal{V}iew\,(V, D)$ is regular and allow for better algorithms. In particular we shall prove in the next sections that the three comparisons we have defined on SAS are all decidable under these restrictions.

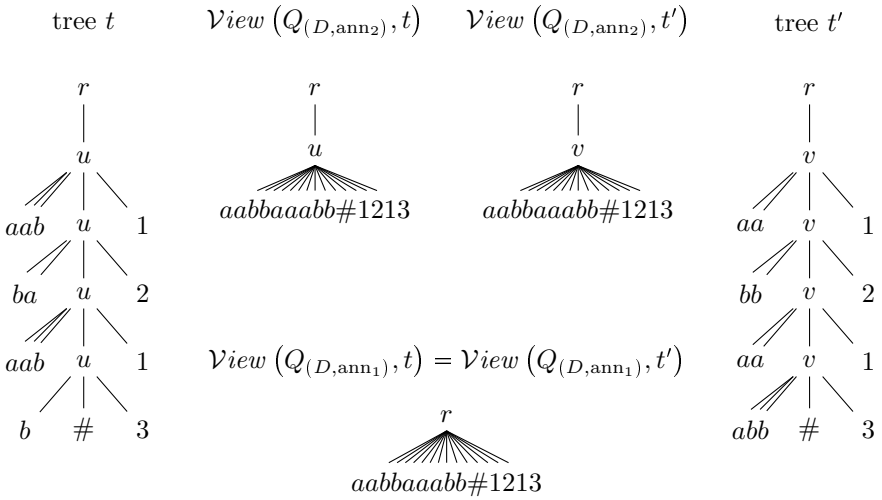


Figure 6: PCP encoding for $\preccurlyeq_3$

Bounded depth. A set of trees L has *bounded depth* if there exists a constant k such that all trees in L have depth at most k. In our setting, it is not the depth of the view trees that we wish to bound, but the depth of the original document. Thus, a query (or view) Q has bounded depth if there exists some k such that every tree in its domain has depth at most k. This implies that $\mathcal{V}iew\,(Q,D)$ has bounded depth, but the latter is not a sufficient condition. For any bounded-depth *MSO* query Q, $\mathcal{V}iew\,(Q,D)$ is clearly a regular set of trees; this can also be viewed as a particular case of Proposition 8. Furthermore, $\mathcal{X}Reg$ and *MSO* clearly have the same expressivity on trees of bounded depth.

Upward closed views. A query (or view) is *upward-closed* if for every document t the parent of every node selected by Q in t is also selected by Q. That means all the ancestors of every visible node are also visible. Equivalently, whenever a node is hidden, all its descendants are hidden as well. For this reason, this requirement is commonly referred to in the literature as the policy's *denial downward consistency* [24] [1].

Interval boundedness. We generalize both bounded depth and upward closed views to allow restricted deletions of internal nodes.

Let t be a tree over $T_{\Sigma\times\{\mathbb{0},\mathbb{1}\}}$. We say that t is *k-interval bounded* if

[1] The term "upward-closed" is employed by Libkin and Sirangelo [21], but a variety of other names appear in the litterature.

1. the label of the root of t belongs to $\Sigma \times \{\mathbb{1}\}$
2. on any descending path of t, there are at most k consecutive nodes with label in $\Sigma \times \{\mathbb{0}\}$ between two nodes with label in $\Sigma \times \{\mathbb{1}\}$.

A tree language $L \subseteq T_{\Sigma\times\{\mathbb{0},\mathbb{1}\}}$ is k-interval bounded if every tree of L is k-interval bounded and an annotation A is *k-interval bounded* if $A(T_\Sigma)$ is k-interval bounded. We say that an annotation A is *interval bounded* (IB) if there exists some k such that A is k-interval bounded. We define k-interval bounded queries and interval bounded queries likewise: query Q is k-interval bounded iff A_Q is. We observe from the definition that any interval bounded query (or annotation) is always root preserving.

Remark 1. Every upward-closed view is 0-interval bounded, and every view with bounded depth k is $(k-1)$-interval bounded.

We state further properties of interval-bounded *MSO* queries after a few illustrative examples.

Example 5. The security view defined by (D_0, X_0) in Example 2 is interval bounded since DTD D_0 is non recursive. It is actually (also) 1-interval bounded. The following DTD D_1 gives informations about the versionning of projects.

```
projects → project*                    prev → version | ε
project → name, version                files → src, bin, doc
version → number, files, license, prev license → free | propr
```

Annotation ann_1 keeps the last version of each project and hides the others. Moreover, it hides all nodes `version`, `files`, `number` (when no explicit rule is given for an element name, its visibility is inherited from its parent):

```
projects → project*                    files → src, bin, doc
project → name, version                 ann1(files, src)
 ann1(project, version) = false         = ann1(files, bin)
version → number, files, license, prev  = ann1(files, doc)
 ann1(version, license)                 = [⇑::files/⇑::version/⇑::project]
 = [⇑::version/⇑::project]             license → free | propr
prev → version | ε
```

The DTD D_1 is recursive but query $Q_{(D_1,\text{ann}_1)}$ is also 1-interval bounded, and $\mathcal{V}iew\left(Q_{(D_1,\text{ann}_1)}, L(D_1)\right)$ is the language validated by the following DTD D_1':

```
projects → project*                    license → free | propr
project → name, src, bin, doc, license
```

The preceding policy is not upward closed as it hides the `version` nodes that are children of the `project` nodes but discloses the `files` children of those hidden `version` nodes. If we replace, however, the annotation ann_1 by ann_1'

defined by the unique mapping $\mathrm{ann}_1(\texttt{version}, \texttt{prev}) = \mathsf{false}$, then the resulting policy is upward-closed (and therefore interval bounded). The corresponding view DTD is D_1' given below:

$\texttt{projects} \to \texttt{project}^*$
$\texttt{project} \to \texttt{name}, \texttt{version}$
$\texttt{version} \to \texttt{number}, \texttt{files}, \texttt{license}$
$\texttt{license} \to \texttt{free} \mid \texttt{propr}$
$\texttt{files} \to \texttt{src}, \texttt{bin}, \texttt{doc}$

Example 6. Let us consider a slightly more complex example: we allow the previous version of project to be a collection of projects. This corresponds to the following case scenario: projects are allowed to merge over time, but not to branch. We define a new DTD D_2 obtained from D_1 by changing the production of `prev` for: $\texttt{prev} \to \texttt{project}^*$. All other production rules remain the same.

Annotation ann_2 keeps licenses together with the name and version of the corresponding project and the project node, and hides every other node.

$\texttt{projects} \to \texttt{project}^*$
$\texttt{project} \to \texttt{name}, \texttt{version}$
$\mathrm{ann}_2(\texttt{project}, \texttt{name}) = \mathsf{false}$
$\mathrm{ann}_2(\texttt{project}, \texttt{version}) = \mathsf{false}$
$\texttt{version} \to \texttt{number}, \texttt{files}, \texttt{license}, \texttt{prev}$
$\mathrm{ann}_2(\texttt{version}, \texttt{license}) = \mathsf{true}$
$\texttt{prev} \to \texttt{project}^*$
$\mathrm{ann}_2(\texttt{prev}, \texttt{project}) = \mathsf{true}$
$\texttt{files} \to \texttt{src}, \texttt{bin}, \texttt{doc}$
$\texttt{license} \to \texttt{free} \mid \texttt{propr}$

The query $Q_{(D_2, \mathrm{ann}_2)}$ is not 1-interval bounded, but it is 2-interval bounded. The corresponding view DTD is D_2 given below:

$\texttt{projects} \to \texttt{project}^*$
$\texttt{project} \to \texttt{name}, \texttt{license}, \texttt{project}^*$
$\texttt{license} \to \texttt{free} \mid \texttt{propr}$

As a last example, suppose we only want to store all licenses without further information. This can be achieved, for instance, via annotation ann_2': $\mathrm{ann}_2'(\texttt{projects}, \texttt{project}) = \mathsf{false}$, and $\mathrm{ann}_2'(\texttt{version}, \texttt{license}) = \mathsf{true}$. The query $Q_{(D_2, \mathrm{ann}_2')}$ is not interval bounded. The resulting view DTD contains a single production rule: $\texttt{projects} \to \texttt{license}^*$.

Below we are stating the main property of interval bounded views, namely, that interval bounded views preserve regularity.

Proposition 8. *For any interval bounded MSO query Q, for any regular language $L \subseteq \mathrm{dom}(Q)$, the language $\mathcal{V}iew\,(Q, L)$ is regular.*

Proof. Let Q be a k-interval-bounded MSO query. It is easy to build a VPA $\mathcal{A} = (\Sigma \times \{\mathbb{0}, \mathbb{1}\}, S_A, \Gamma, I, F, R)$ that recognizes $A_Q(L)$ from VPAs defining Q and recognizing L, because $A_Q(L) = A_Q(T_\Sigma) \cap \Pi_\Sigma^{-1}(L)$. We define the VPA $\mathcal{A}'$ as follows, using $\Gamma^{\leqslant k}$ to denote the union $\bigcup_{0 \leqslant i \leqslant k} \Gamma^i$:

$\mathcal{A}' = (\Sigma, S', \Gamma', I', F', R')$ where

- $S' = S_A \times \Gamma^{\leqslant k}$
- $\Gamma' = \Gamma^{\leqslant k} \times \Gamma$
- $I' = I \times \{\varepsilon\}$,
- $F = F \times \{\varepsilon\}$
- R' is defined as follows
  - $\mathcal{A}'$ has transition $\langle q, w\rangle \xrightarrow{\epsilon} \langle q', w \cdot \Gamma\rangle$ for all transition $q \xrightarrow{(\mathrm{op},(a,\mathbb{0})):\Gamma} q'$ in R, and $w \in \Gamma^{<k}$
  - $\mathcal{A}'$ has transition $\langle q, w \cdot \Gamma\rangle \xrightarrow{\epsilon} \langle q', w\rangle$ for all transition $q \xrightarrow{(\mathrm{cl},(a,\mathbb{0})):\Gamma} q'$ in R, and $w \in \Gamma^{<k}$
  - $\mathcal{A}'$ has transition $\langle q, w\rangle \xrightarrow{(\mathrm{op},a):\langle w,\Gamma\rangle} \langle q', \epsilon\rangle$, for all transition $q \xrightarrow{(\mathrm{op},(a,\mathbb{1})):\Gamma} q'$ in R, and $w \in \Gamma^{\leqslant k}$.
  - $\mathcal{A}'$ has transition $\langle q, \epsilon\rangle \xrightarrow{(\mathrm{cl},a):\langle w,\Gamma\rangle} \langle q', w\rangle$ for all transition $q \xrightarrow{(\mathrm{cl},(a,\mathbb{1})):\Gamma} q'$ in R, and $w \in \Gamma^{\leqslant k}$.
  - $\mathcal{A}'$ has transition $\langle q, w\rangle \xrightarrow{\epsilon} \langle q', w\rangle$ for all $w \in \Gamma^k$ and q, q' such that there is some tree t over alphabet $\Sigma \times \{\mathbb{0}\}$ accepted by the VPA $(\Sigma \times \{\mathbb{0}, \mathbb{1}\}, S_A, q, q', \Gamma, R)$.

We claim that $L(\mathcal{A}') = \mathcal{V}iew\,(Q, L)$.
The last condition corresponds to an epsilon transition from state q to state q' whenever there is some tree t such that the second component of any label in t is $\mathbb{0}$ and some run of the automaton $\mathcal{A}$ can exit from t in state q' if it enters in state q.

When $\mathcal{A}'$ reads an hidden element, it uses an epsilon transition and simulates the stack of $\mathcal{A}$ within its states. Note that this simulation is complete by our hypothesis of interval-boundedness. When opening visible elements, $\mathcal{A}'$ records the information of previous simulations in the stack, so that they may be recovered on the corresponding closing tag. This concludes the proof for Proposition 8. □

The following result is useful to analyze the complexity of our constructions.

Proposition 9. *Let Q be a query given by an automaton $\mathcal{A}$ over $\Sigma \times \{\mathbb{0}, \mathbb{1}\}$ with N states, then Q is interval bounded iff Q is (N^2+1)-IB.*

Proof. Let us suppose that Q is k-IB for some k, but not (N^2+1)-IB. Then there is some tree $t \in \mathrm{dom}(Q)$ such that $t' = A_Q(t)$ is not (N^2+1)-bounded: there is a path in t' from some node n to some of its descendants n' such that $\lambda_{t'}(n)$ and $\lambda_{t'}(n')$ belong to $\Sigma \times \{\mathbb{1}\}$, there are at least (N^2+1) nodes on the path between n and n' and all these nodes between n and n' have label in $\Sigma \times \mathbb{0}$. Since there are at least (N^2+1) such nodes, this implies that on some (as a matter of fact, "on any") accepting run ρ of $\mathcal{A}$ on t', there are two nodes n_1 and n_2 such that $\rho(n_1) = \rho(n_2)$. The usual pumping argument contradicts the interval-boundedness of Q. □

Proposition 10. *For any query Q given by an automaton $\mathcal{A}$ over $\Sigma \times \{\mathbb{0}, \mathbb{1}\}$, testing whether Q is interval bounded is in PTIME.*

PROOF (OUTLINE). Roughly speaking, the set of all k-IB trees can be defined by a deterministic automaton with $O(k)$ states. Hence, it suffices to combine the previous proposition and a simple polynomial algorithm for testing inclusion of tree automata. □

Proposition 11. *Testing whether a query given by a $\mathcal{X}Reg$ expression is interval bounded is* EXPTIME*-complete.*

PROOF. Building an automaton from an $\mathcal{X}Reg$ expression is in EXPTIME (see [6]). Hence, the EXPTIME upper bound follows from Proposition 10. To show EXPTIME-hardness, we reduce satisfiability of $\mathcal{X}Reg$ (see [9, 23]) to testing interval boundedness. Let Q be a $\mathcal{X}Reg$ expression over an alphabet Σ. We define DTD D as follows: $D = (\Sigma \uplus \{a, b\}, r, P)$ where $P'(r) = \Sigma^* a \mid w \in P(r)\}$, $P(a) = a|b$, $P(b) = \varepsilon$ and, for every $\alpha \in \Sigma \backslash \{r\}$, $P(\alpha) = \Sigma^*$. We rewrite Q in linear time into an expression Q' that checks whether the tree satisfies D and whether Q can be satisfied using only the elements from Σ. If those checks succeed, then Q' selects the (unique) node labeled b, and selects no other node except the root, otherwise it selects only the root. Because the DTD D allows to have b elements at arbitrary depth, the view defined by query Q' is interval bounded iff Q is not satisfiable.

Here is how we can build Q'. We denote by Q_0 the expression resulting from the addition of a filter $[\mathsf{not}(\mathsf{self}{::}a \text{ or } \mathsf{self}{::}b)]$ to each elementary axis of Q; every occurrence of $\Rightarrow$, for instance, is replaced by the expression:

$$[\mathsf{not}(\mathsf{self}{::}a \text{ or } \mathsf{self}{::}b)]/\Rightarrow/[\mathsf{not}(\mathsf{self}{::}a \text{ or } \mathsf{self}{::}b)]\,.$$

We also build in linear time an expression Q_D such that for every tree t, $t \models Q_D$ iff $t \in L(D)$. The expression Q' can be built in linear time from Q_D and Q_0:

$$Q' = \Downarrow^*[\mathsf{not} \Uparrow \mathsf{or}\ (\mathsf{self}{::}b \text{ and } \Uparrow^*/[\mathsf{not} \Uparrow \text{ and } Q_0 \text{ and } Q_D])] \qquad \square$$

6. Comparing Security Policies: *MSO*

Reminder: In this section, we assume the query is given by way of an automaton $\mathcal{A}$ over $\Sigma \times \{\mathbb{0}, \mathbb{1}\}$, that recognizes a maximal language.

Notation: Given root preserving annotations A_1 and A_2, for every tree $t = (N_t, root_t, child_t, next_t, \lambda_t)$, we denote by $t \otimes A_1 \otimes A_2$ the tree $t \otimes A_1 \otimes A_2 = (N_t, root_t, child_t, next_t, \lambda'_t)$ such that for all $n \in N_t$, $\lambda'_t(n) = (\lambda_t(n), A_1(n), A_2(n))$.

For interval-bounded annotations, testing comparison 2 amounts to testing determinacy:

Lemma 7. *Let Q_1 and Q_2 denote two MSO queries, with* $\mathrm{dom}(Q_1) = \mathrm{dom}(Q_2)$ *and Q_2 interval-bounded. Then*

- $Q_1 \preccurlyeq_{2,MSO} Q_2$ *iff* $\forall t, t', \mathcal{V}iew\,(Q_2, t) = \mathcal{V}iew\,(Q_2, t') \implies \mathcal{V}iew\,(Q_1, t) = \mathcal{V}iew\,(Q_1, t')$,
- *i.e.,* $Q_1 \preccurlyeq_{2,MSO} Q_2$ *iff* $Q_1 \preccurlyeq_2 Q_2$.

Furthermore, if Q_2 is k-interval-bounded and $Q_1 \preccurlyeq_2 Q_2$, one can compute a query (automaton) Q such that $\mathsf{Rewrite}(Q, Q_2) = Q_1$ *in time exponential in k.*

Proof. Since $Q_1 \preccurlyeq_{2,MSO} Q_2 \implies Q_1 \preccurlyeq_2 Q_2$, all we need is to prove that we can compute a query (automaton) Q such that $\mathsf{Rewrite}(Q, Q_2) = Q_1$ whenever $Q_1 \preccurlyeq_2 Q_2$. Let $k \in \mathbb{N}$ be a natural number such that Q_2 is k-interval-bounded. We suppose $Q_1 \preccurlyeq_2 Q_2$. Then, by Proposition 4, $Q_1 \preccurlyeq_1 Q_2$. We define an automaton $\mathcal{A} = (\Sigma \times \{\mathbb{0}, \mathbb{1}\}^2, S, \Gamma, I, F, R)$ with language $L(\mathcal{A}) = \{t \otimes A_{Q_1} \otimes A_{Q_2} \mid t \in dom(Q_1)\}$. Note that since we suppose $Q_1 \preccurlyeq_1 Q_2$, no label $(a, \mathbb{1}, \mathbb{0})$ can occur in any tree recognized by $\mathcal{A}$. Next, to abstract from elements in t that are not selected by Q_2, in order to 'rewrite' Q_1 in terms of Q_2, we use the same construction as in Proposition 8 which computes an automaton for the view. Indeed, $\mathcal{A}$ can be considered as defining an interval bounded query on trees labeled by $\Sigma \times \{\mathbb{0}, \mathbb{1}\}$ which will select all the nodes labeled by $\Sigma \times \{\mathbb{1}\}$ as $Q_1 \preccurlyeq_1 Q_2$.

Construction of an automaton rewriting Q_1 in terms of Q_2: the idea is to eliminate transitions $q \xrightarrow{(\eta,(a,\mathbb{0},\mathbb{0})):\gamma} q'$ for every $q, q' \in S, \eta \in \{\text{op}, \text{cl}\}, q \in \Sigma, \gamma \in \Gamma$, replacing them with '$\epsilon$' transitions. The interval-boundedness restriction allows us to eliminate those transitions. First, let $\mathcal{E} \subseteq S \times S$ be the set of all pairs (q, q') such that $\mathcal{A}$ accepts some tree with labels in $\Sigma \times \{\mathbb{0}\} \times \{\mathbb{0}\}$ from initial state q to final state q'. More formally, $(q, q') \in \mathcal{E}$ if and only if there is some tree t in $L\,((\Sigma, S, \Gamma, \{q\}, \{q'\}, R))$, with $\lambda_t(n) \in \Sigma \times \{\mathbb{0}\} \times \{\mathbb{0}\}$ for all $n \in N_t$.

We define a VPA $\mathcal{B} = (\Sigma \times \{\mathbb{0}, \mathbb{1}\}, S \times \Gamma^{\leqslant k}, \Gamma \times \Gamma^{\leqslant k}, I \times \{\epsilon\}, F \times \{\epsilon\}, R')$ from $\mathcal{A}$ as follows. Basically, $\mathcal{B}$ simulates within its state a stack of depth at most k.

- $\mathcal{B}$ has transition $(q, u) \xrightarrow{\epsilon} (p, \gamma u)$ for every transition $q \xrightarrow{(\text{op},(a,\mathbb{0},\mathbb{0})):\gamma} p$ of $\mathcal{A}$ and $u \in \Gamma^{\leqslant (k-1)}$.
- $\mathcal{B}$ has transition $(q, \gamma u) \xrightarrow{\epsilon} (p, u)$ for every transition $q \xrightarrow{(\text{cl},(a,\mathbb{0},\mathbb{0})):\gamma} p$ of $\mathcal{A}$ and $u \in \Gamma^{\leqslant (k-1)}$.
- $\mathcal{B}$ has transition $(q, u) \xrightarrow{(\text{op},(a,x_1)):\langle\gamma,u\rangle} (p, \epsilon)$ for every transition $q \xrightarrow{(\text{op},(a,x_1,\mathbb{1})):\gamma} p$ of $\mathcal{A}$ and $u \in \Gamma^{\leqslant k}$.
- $\mathcal{B}$ has transition $(q, \epsilon) \xrightarrow{(\text{cl},(a,x_1)):\langle\gamma,u\rangle} (p, u)$ for every transition $q \xrightarrow{(\text{cl},(a,x_1,\mathbb{1})):\gamma} p$ of $\mathcal{A}$ and $u \in \Gamma^{\leqslant k}$.
- $\mathcal{B}$ has transition $(q, u) \xrightarrow{\epsilon} (p, u)$ for every $u \in \Gamma^{\leqslant k}$ and $(q, p) \in \mathcal{E}$.

One can compute $\mathcal{B}$ from $\mathcal{A}$ in time exponential in k. There is a polynomial p_1 such that $|\mathcal{B}| \leqslant (p_1(|Q_1| \times |Q_2|))^k$. To conclude the proof, we observe that due to our determinacy hypothesis, $\mathcal{B}$ recognizes a maximal language, and by

construction, it defines a query Q such that $\mathsf{Rewrite}(Q, Q_2) = Q_1$, as evidenced by the following invariant.

Let w a word over $\{\mathrm{op}, \mathrm{cl}\} \times \Sigma$, (q, u) a state in $S \times \Gamma^{\leqslant k}$, and σ a word over $\Gamma \times \Gamma^{\leqslant k}$. We denote by σ' the same σ considered as a word over Γ. We also define for every word w' over $\{\mathrm{op}, \mathrm{cl}\} \times \Sigma \times \{\mathbb{0}, \mathbb{1}\} \times (\Sigma \cup \{\mathbb{0}\})$ the word $\pi_{2,3}(w')$ as the word obtained from w by projecting each letter on its second and third component, and then removing all occurrences of letter $(\mathbb{0}, \mathbb{0})$. We claim that for all such w, q, and σ, $\mathcal{B}$ preserves the following invariant.

Invariant: *$\mathcal{B}$ can reach configuration $((q, u), \sigma)$ after reading w if and only if there exists a word w' over $\{\mathrm{op}, \mathrm{cl}\} \times \Sigma \times \{\mathbb{0}, \mathbb{1}\} \times (\Sigma \cup \{\mathbb{0}\})$ such that the following two conditions are satisfied: (1) $\pi_{2,3}(w') = w$, and (2) $\mathcal{A}$ can reach configuration $(q, \sigma' u)$ after reading w'.* □

From this, since determinacy is co-recursively enumerable, and $\preccurlyeq_2$ is recursively enumerable, we can deduce immediately the decidability of $\preccurlyeq_2$ for interval-bounded annotations, but we can do much better. A first approach for testing $\preccurlyeq_2$ could be to build the "square" of $\mathcal{B}$ and test whether there are two trees $t \neq t'$ accepted by $\mathcal{B}$, with the same projection over Σ; $\Pi_\Sigma(t) = \Pi_\Sigma(t')$. We would be able to test this property on $\mathcal{B}$, in terms of accessibility of states.

Corollary 1. *Let Q_1 and Q_2 be two MSO queries, such that $\mathrm{dom}(Q_1) = \mathrm{dom}(Q_2)$. Given a fixed constant k, if Q_2 is k-interval bounded, then we can test in polynomial time whether $Q_1 \preccurlyeq_2 Q_2$ This holds in particular for downward closed views.*

Similarly, when the depth of the domain is bounded by a fixed constant, the complexity for testing $Q_1 \preccurlyeq_2 Q_2$ becomes NLogspace.

Proof. We first check in polynomial time that $Q_1 \preccurlyeq_1 Q_2$; otherwise, $Q_1 \not\preccurlyeq_2 Q_2$. We then build the automaton $\mathcal{B}$ above, and eliminate its epsilon transitions, resulting in a VPA $(\Sigma \times \{\mathbb{0}, \mathbb{1}\}, S_B, \Gamma_B, I_B, F_B, R_B)$. Finally, we build the square $\mathcal{B}_{square}$ of this automaton $\mathcal{B}$, namely $(\Sigma \times \{\mathbb{0}, \mathbb{1}\} \times \{\mathbb{0}, \mathbb{1}\}, S_B^2, \Gamma_B^2, I_B^2, F_B^2, R_{square})$ such that $\mathcal{B}_{square}$ has rule $(q_1, q_2) \xrightarrow{(\eta,(b,\alpha_1,\alpha_2)):(\gamma_1,\gamma_2)} (q_1', q_2') \in R_{square}$ iff $\mathcal{B}$ has rules $q_1 \xrightarrow{(\eta,(b,\alpha_1)):\gamma_1)} q_1' \in R$ and $q_2 \xrightarrow{(\eta,(b,\alpha_2)):\gamma_2)} q_2' \in R$. By construction, and as we supposed $Q_1 \preccurlyeq_1 Q_2$, it holds that $Q_1 \preccurlyeq_2 Q_2$ if and only if for all b, α_1, α_2 with $\alpha_1 \neq \alpha_2$, the language of $\mathcal{B}_{\text{square}}$ contains no tree with a node labeled (b, α_1, α_2). This is a problem of reachability, which can be solved in polynomial time for VPAs. Since there is a polynomial p_2 such that $\mathcal{B}_{square}$ is built in time at most $\left(p_2(|Q_1| \times |Q_2|)\right)^k$, we get the polynomial time complexity when k is a fixed constant.

When the depth of the domain is bounded by a fixed constant k, we observe that the stack of any run of $\mathcal{B}_{square}$ over a tree of depth k can be represented as a word in $(\Gamma_B^2)^{\leqslant k}$. Therefore, $\mathcal{B}_{square}$ is equivalent to a word automaton $\mathcal{A}_w$ of polynomial size. We cannot afford to build the full $\mathcal{B}_{square}$, and even less $\mathcal{A}_w$, but we can simulate $\mathcal{A}_w$ on-the fly: each transition can be simulated using logarithmic space and if $\mathcal{A}_w$ accepts a word containing some label (b, α_1, α_2)

with $\alpha_1 \neq \alpha_2$, then it accepts such a tree of size polynomial, which gives a NLogspace algorithm for testing $Q_1 \preccurlyeq_2 Q_2$.

Observe that it is actually sufficient to assume that Q_2 is k-interval bounded or has domain of bounded depth. Query Q_1 need not be constrained. □

However, the full construction of $\mathcal{B}$ induces an exponential cost in terms of time and space, so that for general interval-bounded queries, this approach uses exponential time. We provide a polynomial space algorithm instead for interval-bounded queries.

Lemma 8. *Let Q_1, Q_2 be MSO queries, with Q_2 interval bounded. If there are two trees t, t' such that $\mathcal{V}iew\,(Q_2, t) = \mathcal{V}iew\,(Q_2, t')$ but $\mathcal{V}iew\,(Q_1, t) \neq \mathcal{V}iew\,(Q_1, t')$, then there are two such trees of size exponential and depth polynomial in the size of the automata Q_1, Q_2.*

Proof. We prove this with a pumping argument, adapting the standard pumping arguments for tree automata in order to preserve the difference between the views for Q_1 (we shall therefore consider three nodes instead of two). Let $\mathcal{A}_1 = (\Sigma, S_1, \Gamma_1, I_1, F_1, \Delta_1)$ and $\mathcal{A}_2 = (\Sigma, S_2, \Gamma_2, I_2, F_2, \Delta_2)$ be two query automata, with corresponding queries Q_1 and Q_2 such that Q_2 is an interval bounded query and $Q_1 \preccurlyeq_1 Q_2$. Let (t, t') be a pair of trees of minimal size such that $\mathcal{V}iew\,(Q_2, t) = \mathcal{V}iew\,(Q_2, t')$ but $Q_1(t) \neq Q_1(t')$. Let ρ_2^t (resp. $\rho_2^{t'}$) denote accepting runs of the automaton $\mathcal{A}_2$ on $\mathcal{A}_{Q_2}(t)$ (resp. $\mathcal{A}_{Q_2}(t')$), and ρ_1^t (resp. $\rho_1^{t'}$) denote accepting runs of the automaton $\mathcal{A}_1$ on $\mathcal{A}_{Q_1}(t)$ (resp. $\mathcal{A}_{Q_1}(t')$). We also denote by $(\rho_2^t)^\uparrow$, $(\rho_1^{t'})^\uparrow$, etc. the corresponding functions that map a node n to the pair of states assigned by the run of the automaton before reading the opening tag and after processing the closing tag of n.

Vertical pumping: We decorate every node n in $Q_2(t)$ (therefore also in $Q_2(t')$) with the tuple $\rho(n) = (\rho_2^t(n), \rho_2^{t'}(n), \rho_1^t(n), \rho_1^{t'}(n))$. Suppose there is some node in $Q_2(t)$ at depth strictly greater than $(k+1) \times 2 \times |S_2|^2 \times |S_1|^2$ in t or t', then there are three distinct nodes $n^\uparrow, n^\circ, n^\downarrow$ in $Q_2(t)$ such that $n^\uparrow$ is an ancestor of n°, n° an ancestor of $n^\downarrow$, and $\rho(n^\uparrow) = \rho(n^\circ) = \rho(n^\downarrow)$ as depicted in Figure 7.

We consider two cases depending on whether there exists below n° a node n that belongs to $Q_1(t) \,\Delta\, Q_1(t')$. In the first case we assume there is some node n below n° that belongs to $Q_1(t) \,\Delta\, Q_1(t')$. Then we could replace the subtree below $n^\uparrow$ with the subtree below n° in t and t': the two trees thus obtained would have same view for Q_2 and different views for Q_1, which contradicts minimality of the pair (t, t'). In the second case there is no node $n \in Q_1(t) \,\Delta\, Q_1(t')$ below n°, but then we could replace the subtree below n° with the subtree below $n^\downarrow$ in t and t': the two trees thus obtained would have same view for Q_2 and different views for Q_1, which contradicts minimality of the pair (t, t'). So either way, our minimality hypothesis enters in contradiction with the existence of a node of depth greater than $(k+1) \times 2 \times |S_2|^2 \times |S_1|^2$ in $Q_2(t)$ or in $Q_2(t')$. Hence no node in $Q_2(t)$ or $Q_2(t')$ has depth greater than $(k+1) \times 2 \times |S_2|^2 \times |S_1|^2$.

Thus, t and t' have polynomial depth. Notice that the pumping argument used to bound the depth of the trees does not increase the size of the trees.

We can use another pumping argument, pumping "horizontally" this time, and bound the number of children of every node in t or t' by an exponential.

Horizontal pumping: As before we use a pumping argument over nodes in $Q_2(t)$, because this makes it easier to preserve equality of the views for Q_2. Let $n \in Q_2(t)$. Then n also belongs to $Q_2(t')$. However, it could very well be that no child of n in t or t' belongs to $Q_2(t)$, while some descendant of n would still belong to $Q_2(t)$. To avoid those difficulties, we consider the children $n_1, n_2, \ldots, n_M$ of n in $\mathcal{V}iew\,(Q_2, t)$, in document order. We decorate each node n_i with two tuples $\vec{\rho}(n_i, \mathrm{op})$ and $\vec{\rho}(n_i, \mathrm{cl})$ in $(S_1 \times \Gamma_1^{\leqslant k})^2 \times (S_2 \times \Gamma_2^{\leqslant k})^2$. Tuple $\vec{\rho}(n_i, \mathrm{op})$ is associated to the opening tag of n_i and $\vec{\rho}(n_i, \mathrm{cl})$ to its closing tag. The tuples are defined as follows. Let $d_t \leqslant k$ denote the number of stack symbols that have been added (and not yet removed) after reading the opening tag of n and before reading the opening tag of n_i in t: $d_t(n_i) = \mathit{depth}_t(n_i) - \mathit{depth}_t(n) - 1$, and similarly $d_{t'}(n_i) = \mathit{depth}_{t'}(n_i) - \mathit{depth}_{t'}(n) - 1$. The tuples $\vec{\rho}(n_i, \mathrm{op})$ and $\vec{\rho}(n_i, \mathrm{cl})$ are respectively defined as $((q_2, u_2), (q_2', u_2'), (q_1, u_2), (q_1', u_1'))$ and $((s_2, u_2), (s_2', u_2'), (s_1, u_2), (s_1', u_1'))$ where $(\rho_2^t)^{\uparrow}(n_i) = (q_2, s_2)$, $(\rho_1^{t'})^{\uparrow}(n_i) = (q_1', s_1')$, etc. and $u_2 \in (\Gamma_2)^{d_t(n_i)}$ contains the $d_t(n_i)$ topmost symbols of the stack for run ρ_2^t before processing the opening tag of node n_i, $u_1' \in (\Gamma_1)^{d_{t'}}$ contains the $d_{t'}(n_i)$ topmost symbols of the stack for run $\rho_1^{t'}$ before processing the opening tag of node n_i etc.

We assume that Γ_1, Γ_2 both contain at least two elements. The other cases can be treated similarly. The number of different tuples $\vec{\rho}$ that can be constructed is strictly smaller than $|S_1|^2 \times |\Gamma_1|^{2k+2} \times |S_2|^2 \times |\Gamma_2|^{2k+2}$. Hence if $M \geqslant 2|S_1|^2 \times |\Gamma_1|^{2k+2} \times |S_2|^2 \times |\Gamma_2|^{2k+2}$, there exist $1 \leqslant i < j < l \leqslant M$ such that $\vec{\rho}(n_i, \mathrm{op}) = \vec{\rho}(n_j, \mathrm{op}) = \vec{\rho}(n_l, \mathrm{op})$. This however contradicts the minimality of t and t': the trees $t_{i,j}$ and $t'_{i,j}$ obtained from t and t' by removing all tags between the opening of n_i (included) and the opening of n_j (excluded) satisfy $\mathcal{V}iew\,(Q_2, t_{i,j}) = \mathcal{V}iew\,\big(Q_2, t'_{i,j}\big)$, and likewise the trees $t_{j,l}$ and $t'_{j,l}$ obtained by removing all tags between n_j and n_l. The contradiction stems from the observation that $Q_1(t_{i,j}) \neq Q_1(t'_{i,j})$ or $Q_1(t_{j,l}) \neq Q_1(t'_{j,l})$. This concludes the proof that every node from $Q_2(t)$ has at most $2 \times |S_1|^2 \times |\Gamma_1|^{2k+1} \times |S_2|^2 \times |\Gamma_2|^{2k+1}$ children in $\mathcal{V}iew\,(Q_2, t)$.

We still have to bound the number of nodes in $N_t \backslash Q_2(t)$ and likewise in t', but here the pumping argument is the usual one, as we can apply the pumping argument from Lemma 3 independently in t and t' on the "hidden" parts, provided nodes selected by Q_2 are not affected. For each node $n \in N_t$ and every sequence $n_1, n_2, \ldots, n_L$ of consecutive children of n, if $L \geqslant |S_1| \times |S_2|$ then one of these children has necessarily a descendant in $Q_2(t)$ otherwise the pumping argument from Lemma 3 would contradict the minimality of t and t'. Consequently, the number of children of a node in t or t' can be roughly bounded by $O(|S_1|^3 \times |\Gamma_1|^{2k+2} \times |S_2|^3 \times |\Gamma_2|^{2k+2})$. Moreover, each node $n \in N_t$ without descendant in $Q_2(t)$ has no descendants of depth greater than its own depth plus $|S_1|^2 \times |S_2|^2$, according to the pumping argument of Lemma 3. Therefore, no node in t or t' has depth greater than $k \times 3 \times |S_2|^2 \times |S_1|^2$. The combination of those horizontal and vertical pumping arguments allows to conclude the proof

for Lemma 8: t and t' have size at most exponential. □

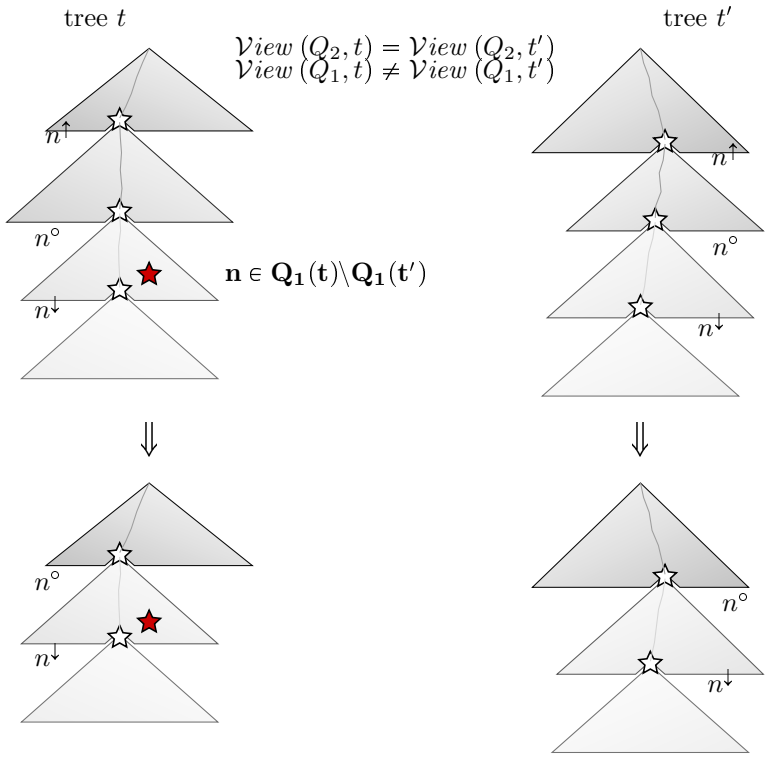


Figure 7: Pumping argument for $\preccurlyeq_2$

Corollary 2. *Given MSO queries Q_1 and Q_2, with Q_2 interval bounded, we can test $Q_1 \preccurlyeq_{2,MSO} Q_2$ in polynomial space.*

Proof. Let Q_1 be an MSO query and Q_2 an MSO k-interval-bounded query. Then, by Lemma 7 it is enough to test whether there are trees t, t' such that $Q_2(t) = Q_2(t')$ but $Q_1(t) \neq Q_1(t')$. Moreover, Lemma 8 gives a bound on the size and depth of t and t'. This suggests the following algorithm: we guess the size of t, t', and guess step by step the run of both view automata over t and t'. We only need to store the stack and the current state, which provides a non-deterministic algorithm in polynomial space. The result then follows from Savitch's theorem. □

In the following, we are interested in MSO queries whose domain can be expressed by a non recursive DTD. We write that *the domain is a non-recursive DTD* even if no DTD is manipulated here. Since queries whose domain a non-recursive DTD are a special case of interval-bounded queries, we get immediately from Corollary 2:

Corollary 3. *Let Q_1 and Q_2 be two MSO queries. When the domain is a non-recursive DTD, one can test $Q_1 \preccurlyeq_{2,MSO} Q_2$ in polynomial space.*

In order to establish the complexity of $Testcomp_{2,MSO}$, we use a reduction from the *Compressed Membership Problem for regular expressions with squares.*

The syntax of regular expression with squares is $E ::\equiv \mathsf{empty} \mid a \mid E\text{","}E \mid E\text{"|"}E \mid E^* \mid E^2$, where E^2 represents E, E. A *straight line program* is a context free grammar $G = (V, T, S, P)$ with V the non-terminals, T the terminals, S the initial non-terminal, and $P : V \to (V \cup T)^*$ the productions, such that there is a single production from each non-terminal, and the production relation is acyclic. Thus, each straight line program G represents a single word w_G. In that setting, the *Compressed Membership Problem* is the problem deciding given a regular expression with squares E (over alphabet T), and a word w over T given by a straight line program, whether w belongs to the language of E.

Theorem 5 (Theorem 6 in [22]). *The Compressed Membership Problem is* PSPACE*-complete for regular expressions with squares.*

Lemma 9. *For MSO queries given by automata, $Testcomp_{2,MSO}$ is* PSPACE*-hard even when the domain is a non-recursive DTD.*

PROOF. The proof works by reduction from the compressed membership problem for regular expressions with squares. Fix a straight line program $G = (V, T, S, P)$ and a regular expression with squares E over T. We can compute in polynomial time a visibly pushdown automaton $\mathcal{A}$ recognizing the derivation trees of G, and another one $\mathcal{A}_E$ whose frontier (the language formed by the leaves of the trees in $L(\mathcal{A}_E)$) is the language of E. Furthermore, $L(\mathcal{A}_E)$ and $L(\mathcal{A})$ can be described by non recursive DTDs.

Let D be the domain that consists of trees with root r, and a unique subtree either in $L(\mathcal{A}_E)$ or in $L(\mathcal{A})$ below the root. Let Q_1, Q_2 be queries over D such that

- Q_1 selects all the leaves of t if t consists of a root r and a subtree in $L(\mathcal{A})$: (then $\mathcal{V}iew\,(Q_1, t)$ represents the word w_G), or selects nothing but the root r if t consists of a root r and a subtree in $L(\mathcal{A}_E)$.
- Q_2 selects all the leave nodes, irrespective of whether they belong to $L(\mathcal{A})or L(\mathcal{A}_E)$.

$Q_1 \preccurlyeq_2 Q_2$ iff w_G does not belong to the language of E. This concludes the proof. □

We can conclude from Corollary 2 and Lemma 9 that

Theorem 6. *$Testcomp_{2,MSO}$ is* PSPACE*-complete when queries are given by automata and the domain is a non-recursive DTD.*

Theorem 7. *$Testcomp_{2,MSO}$ is* PSPACE*-complete for interval-bounded MSO queries given by automata.*

Theorem 8. *$Q_1 \preccurlyeq_3 Q_2$ is* PSPACE*-complete for MSO queries given by automata, when the domain has bounded depth k.*

PROOF. We have the hardness by using the same construction as in Lemma 9. Let us prove that this problem can be decided in polynomial space.

Here is a proof following a schema similar to $\preccurlyeq_2$: we define an automaton $\mathcal{A} = (\Sigma \times \{\mathbb{0}, \mathbb{1}\}^2, S, \Gamma, I, F, R)$ with language $L(\mathcal{A}) = \{t \otimes A_{Q_1} \otimes A_{Q_2} \mid t \in L(D)\}$.

We transform $\mathcal{A}$ into a word transducer from $\mathcal{V}iew\,(Q_2, -)$ to $\mathcal{V}iew\,(Q_1, -)$. We build a word automaton $\mathcal{A}_w = (\Sigma \times \{\mathbb{0}, \mathbb{1}\}^2, S \times \Gamma^k, I \times \{\varepsilon\}, F \times \{\varepsilon\}, R_w)$ equivalent to $\mathcal{A}$: for all $\eta \in \Sigma \times \{\mathbb{0}, \mathbb{1}\}^2$, $u \in \Gamma^{\leqslant (k-1)}$, $q, q' \in S$, and all $\gamma \in \Gamma$, $\mathcal{A}_w$ has rule $(q, u) \xrightarrow{(\mathrm{op},\eta)} (q', u\gamma)$ iff $\mathcal{A}$ has rule $q \xrightarrow{(\mathrm{op},\eta):\gamma} q'$. $\mathcal{A}_w$ has rule $(q, u\gamma) \xrightarrow{(\mathrm{cl},\eta)} (q', u)$ iff $\mathcal{A}$ has rule $q \xrightarrow{(\mathrm{cl},\eta):\gamma} q'$.

From $\mathcal{A}_w$ we build automaton $\mathcal{B}_w = (\Sigma \times \{\mathbb{0}, \mathbb{1}\}^2, S \times \Gamma^k, I \times \{\varepsilon\}, F \times \{\varepsilon\}, R_B)$, such that for all $x_1, x_2 \in \{\mathbb{0}, \mathbb{1}\}$, $u \in \Gamma^{\leqslant (k-1)}$, $q, q' \in S$, and all $\gamma \in \Gamma$, $\mathcal{B}_w$ has rule $(q, u) \xrightarrow{(\mathrm{op},x_1,x_2)} (q', u\gamma)$ iff $x_1 = \mathbb{1}$ or $x_2 = \mathbb{1}$ and there exists $b \in \Sigma$ such that $\mathcal{A}_w$ has rule $(q, u) \xrightarrow{(\mathrm{op},(b,x_1,x_2))} (q', u\gamma)$. $\mathcal{B}_w$ has rule $(q, u) \xrightarrow{\varepsilon} (q', u\gamma)$ iff there exists $b \in \Sigma$ such that $\mathcal{A}_w$ has rule $(q, u) \xrightarrow{(\mathrm{op},(b,\mathbb{0},\mathbb{0}))} (q', u\gamma)$. We add similar rules for the closing tags. We remark that the number of consecutive transitions in a minimal (accepting) run of $\mathcal{B}_w$ over some input is bounded by $|\mathcal{A}_w|^k$.

Now, we can see $\mathcal{B}_w$ as a word transducer of polynomial size (remember that k is a fixed constant), and $Q_1 \preccurlyeq_2 Q_2$ if and only if that transducer is functional. We use the result from [17, 18] that functionality of word transducers is decidable in NLOGSPACE. Their proof uses result on the emptiness of automata with reversal-bounded counters to prove that whenever there is an input on which a word transducer T can produce two different outputs then there is such an input of size polynomial in T. Here, $\mathcal{B}_w$ is of exponential size, so that we cannot afford to build it, but we can simulate its transitions on-the-fly, and check for every input v of size polynomial in $|\mathcal{B}_w|$ – i.e., for every input of exponential size – if $\mathcal{B}_w$ can produce two different outputs on v. This gives a non-deterministic algorithm in polynomial space: guess the size of the input, and simulate $\mathcal{B}_w$ on-the-fly on this input. The result then follows from Savitch's theorem. □

Theorem 9. *$Testcomp_{3,MSO}$ is in* EXPTIME *for interval-bounded queries when queries are given by automata.*

PROOF. See Appendix

7. Comparing Security Policies: $\mathcal{X}Reg$

In this section we suppose the queries Q_1 and Q_2 are given by $\mathcal{X}Reg$ expressions. Query containment is EXPTIME-complete[23] for $\mathcal{X}Reg$ and since this holds for boolean queries, the interval-boundedness restriction does not help. This complexity can be lowered to PSPACE over non-recursive DTDs.

Theorem 10. *Let $Q_{\mathcal{X}}$ and $Q_{\mathcal{X}'}$ be two root preserving $\mathcal{X}Reg$ queries. When the domain of $Q_{\mathcal{X}}$ is a non-recursive DTD, deciding $Q_{\mathcal{X}} \preccurlyeq_1 Q_{\mathcal{X}'}$ is* PSPACE*-complete.*

PROOF. For $\mathcal{X}Reg$, query containment and satisfiability are equivalent problems: $Q_{\mathcal{X}} \preccurlyeq_1 Q_{\mathcal{X}'}$ if and only if $(\mathcal{X}^{-1}/[\neg\Uparrow]) \wedge \neg(\mathcal{X}'^{-1}/[\neg\Uparrow])$ is not satisfiable. The PSPACE-hardness is obvious since $\mathcal{X}Reg$ generalize regular expressions, and containment for regular expressions is PSPACE-hard. Moreover, $\mathcal{X}Reg$ has the small model property: it is a well known property of PDL, hence of $\mathcal{X}Reg$ [3] that for every $\mathcal{X} \in \mathcal{X}Reg$, if there exists a tree t that satisfies $\mathcal{X}$, then there exists such a tree of size at most exponential in $\mathcal{X}$. As a consequence of this, the PSPACE-completeness of query containment for $\mathcal{X}Reg$ over non-recursive DTDs is not really surprising: [3] identifies the possibility of building exponentially deep models as the main reason why PDL (hence $\mathcal{X}Reg$) is EXPTIME-hard. It is therefore natural that classical model-theoretic methods provide a PSPACE algorithm when we bound the depth of the trees: we build a tree non-deterministically one branch at a time and check that it satisfies $\mathcal{X}$.

[6] provides a linear-time algorithm to translate a $\mathcal{X}Reg$ formula $\mathcal{X}$ into a two-way alternating automaton $\mathcal{A}$. $\mathcal{A}$ has state space S -roughly corresponding to the subformula of $\mathcal{X}$- of linear size, of course. This alternating automaton $\mathcal{A}$ is translated in exponential time into a standard tree automaton $\mathcal{A}'$ such that for every tree t, $\mathcal{A}'$ accepts t if and only if $\mathcal{X}$ is satisfied in t. Actually, the construction in [6] works over first-child-next-sibling encoding, but the translation to VPA is obvious, using [16] for instance, so that we can assume A' to be a VPA $(\Sigma, Q, \Gamma, I, F, R)$.

We cannot afford the full construction of $\mathcal{A}'$, because it induces an exponential cost. However, the states and stack symbols of A' are of the form $\Gamma, Q \in \{0,1\}^{(S\times S\times S)}$, so that representing a state uses only polynomial space. Furthermore, although this result is not mentionned in [6], a careful analysis of the rules for constructing R guarantees that we can check the transitions of $\mathcal{A}'$ using only polynomial time: in the VPA setting, this means that for every $q, q' \in Q$, $\gamma \in \Gamma$, $\eta \in \{\mathrm{op}, \mathrm{cl}\}$ and $a \in \Sigma$, we can check that $q \xrightarrow{(\eta,a):\gamma} q'$ belongs to R using polynomial time.

We non-deterministically guess letter by letter the linearization of the tree and the rule we apply. We only need to remember the stack of the automaton, which is of polynomial size by our hypothesis that the DTD is non-recursive. The result then follows from Savitch's theorem. □

Theorem 11. *Let Q_1 and Q_2 be two root preserving $\mathcal{X}Reg$ queries. When the domain of Q_1 is a non-recursive DTD, deciding $Q_1 \preccurlyeq_{2,\mathcal{X}Reg} Q_2$ is* PSPACE*-complete.*

PROOF. Since *MSO* and $\mathcal{X}Reg$ have the same expressivity when the depth of the trees is bounded, $Q_1 \preccurlyeq_{2,\mathcal{X}Reg} Q_2$ if and only if $Q_1 \preccurlyeq_{2,MSO} Q_2$. So, by Lemma 7, $Q_1 \preccurlyeq_{2,\mathcal{X}Reg} Q_2$ if and only if $Q_1 \preccurlyeq_2 Q_2$. To begin, we first check that $Q_1 \preccurlyeq_1 Q_2$, in polynomial space by Theorem 10. Using the method in [6]

we can build in exponential time two automata $\mathcal{A}_1$ and $\mathcal{A}_2$ over $\Sigma \times \{\mathbb{0}, \mathbb{1}\}$ such that $\mathcal{A}_1$ (resp. $\mathcal{A}_2$) recognizes the language A_{Q_1} (resp. A_{Q_2}). Then, we use a pumping argument similar to Lemma 8: if there are two trees t, t' such that $Q_2(t) = Q_2(t')$ but $Q_1(t) \neq Q_1(t')$, then there are two such trees in which the number of children of every node is a polynomial in $\mathcal{A}_1$ and $\mathcal{A}_2$. Thus, there exists a polynomial p such that the number of children below each node is at most $2^{p(n)}$ where n is the sum of the size of Q_1 and Q_2. Since our hypothesis on the domain bounds the depth of the trees by n, the size of t and t' is at most $2^{p(n)\times n}$. To sum up, we have proved that if there are two trees t, t' such that $Q_2(t) = Q_2(t')$ but $Q_1(t) \neq Q_1(t')$, then there are two such trees of size at most exponential in Q_1 and Q_2.

We cannot afford to build automata $\mathcal{A}_1$ and $\mathcal{A}_2$, but we can simulate their execution on-the-fly: we guess non-deterministically and letter by letter two trees t and t' over $\Sigma \times \{\mathbb{0}, \mathbb{1}\}$ of size exponential in Q_1 and Q_2, and simulate the execution of $\mathcal{A}_1$ and $\mathcal{A}_2$, checking $Q_2(t) = Q_2(t')$ and $Q_1(t) \neq Q_1(t')$. □

We denote by $Memb^{MSO}_{\mathcal{X}Reg}$ the problem of deciding, given a query automaton $\mathcal{QA}$, whether there exists a $\mathcal{X}Reg$ query Q equivalent to $\mathcal{QA}$. Using product alphabet similarly to the proof of Theorem 11, it is obvious that this problem can be reduced in polynomial time to the Boolean version of the problems which we will therefore also denote by $Memb^{MSO}_{\mathcal{X}Reg}$:

Input: A VPA $\mathcal{A}$
Question: *Is there a $\mathcal{X}Reg$ filter f such that for every tree t, $(t, root_t) \models f$ if and only if $t \in L(\mathcal{A})$?*

Proposition 12. *The problem of deciding $\leqslant_{2,\mathcal{X}Reg}$ for interval bounded $\mathcal{X}Reg$ queries can be reduced in exponential time to $Memb^{MSO}_{\mathcal{X}Reg}$.*

Proof. Immediate from the construction in Lemma 7 : We compute an automaton $\mathcal{A}$ with language $L(\mathcal{A}) = \{t \otimes A_{Q_1} \otimes A_{Q_2} \mid t \in dom(Q_1)\}$, test $Q_1 \leqslant_2 Q_2$ and in this case the construction provides a query Q satisfying $\mathsf{Rewrite}(Q, Q_2) = Q_1$. These tests and the construction of Q require at most exponential time. Then, $Q_1 \leqslant_{2,\mathcal{X}Reg} Q_2$ if and only if there exists a $\mathcal{X}Reg$ query equivalent to Q. □

However, since the exact complexity, or even the decidability of problem $Memb^{MSO}_{\mathcal{X}Reg}$ have not been established in the litterature (to the best of our knowledge), this is of little help. Actually, the gap in expressiveness between MSO and $\mathcal{X}Reg$ has been established very recently [31]. Thus, the following result sheds a new light on the problem of deciding $\leqslant_{2,\mathcal{X}Reg}$.

Proposition 13. *$Memb^{MSO}_{\mathcal{X}Reg}$ can be reduced in polynomial time to $\leqslant_{2,\mathcal{X}Reg}$ with interval bounded $\mathcal{X}Reg$ annotations*

Proof. Fix $\mathcal{A} = (\Sigma, Q, \Gamma, I, F, R)$ a VPA, which we assume w.l.o.g. to be complete. That is, we assume $\mathcal{A}$ has a run (not necessarily accepting, of course)

over all trees t in T_Σ. We build a DTD D and interval-bounded queries Q_1, Q_2 defined by $\mathcal{X}Reg$ expressions, such that $Q_1 \preccurlyeq_{2,\mathcal{X}Reg} Q_2$ iff there exists a $\mathcal{X}Reg$ filter f such that for every tree t, $(t, root_t) \models f$ if and only if $t \in L(\mathcal{A})$. We assume without loss of generality that $\Sigma \cap Q = \varnothing$. We build a DTD D over alphabet $\Sigma \cup Q$ defined via the following rules. Abusing notations for regular expressions, we use sets, writing S instead of $s_1 \mid s_2 \mid \ldots \mid s_n$ for a set S consisting of elements $s_1, \ldots s_n$. For all $a \in \Sigma$, $a \to (Q, \Sigma)^*, Q$.

The proof works as follows: under r, D simulates a run of automaton $\mathcal{A}$ over a tree. Q_1 checks the simulation of the transitions and, when the run is valid and leads to an accepting state, Q_1 selects all nodes from the tree with label in Σ. A contrario, if either the run leads to rejection, or if the elements labeled in Σ simulate no valid run, Q_1 selects only the root. Q_2 selects all nodes from the tree with label in Σ when the run is valid, whether it is accepting or it leads to rejection, but selects only the root if the elements labeled in Σ simulate no valid run. The crux of the proof is to make sure with nodes labeled in Q that $\mathcal{V}iew\,(Q_1, D) = L(\mathcal{A})$, while $\mathcal{V}iew\,(Q_2, D)$ is the set of all trees over Σ.

This result is obtained with the following queries: let E be the set of all $(q_1, q_1', q_2, q_2', a)$ in $Q^4 \times \Sigma$ such that there exists some γ in Γ that verifies simultaneously $q_1 \xrightarrow{(\mathrm{op},a):\gamma} q_1'$ and $q_2' \xrightarrow{(\mathrm{cl},a):\gamma} q_2$. We define auxiliary $\mathcal{X}Reg$ filters: $f_\Sigma = \bigvee_{b\in\Sigma} \mathsf{self}{::}b$

$$f_{root} = \left(\bigvee_{q_i \in I} [\Downarrow[\mathsf{not} \Leftarrow]]/\mathsf{self}{::}q_i\right) \wedge \left(\bigvee_{q_f \in F} [\Downarrow[\mathsf{not} \Rightarrow]]/\mathsf{self}{::}q_f\right)$$

$$f^{q_1,q_2}_{q_1',q_2'} = (\mathsf{self}{::}a) \wedge (\Leftarrow{::}q_1) \wedge (\Rightarrow{::}q_2) \wedge \left(\Downarrow[\mathsf{not} \Leftarrow]/\mathsf{self}{::}q_1'\right) \wedge \left(\Downarrow[\mathsf{not} \Rightarrow]/\mathsf{self}{::}q_2'\right)$$

$$f_{valid} = \left[\mathsf{not}\left(\Downarrow^*/\left[f_\Sigma \wedge \left(\mathsf{not} \bigvee_{(q_1,q_1',q_2,q_2',a)\in E} f^{q_1,q_2}_{q_1',q_2'}\right)\right]\right)\right]$$

The two $\mathcal{X}Reg$ queries are defined as $Q_2 = [f_{valid}]/\Downarrow^*/[f_\Sigma] \cup \mathsf{self}[\mathsf{not} \Uparrow]$ and $Q_1 = [f_{valid} \wedge f_{root}]/\Downarrow^*/[f_\Sigma] \cup \mathsf{self}[\mathsf{not} \Uparrow]$. It should be clear that $Q_1 \preccurlyeq_{2,\mathcal{X}Reg} Q_2$ if and only if there exists a $\mathcal{X}Reg$ filter f such that for every tree t, $(t, root_t) \models f$ if and only if $t \in L(\mathcal{A})$. Actually, the two queries Q_1 and Q_2 are even downward-closed. □

From this proof and the expressivity gap between MSO and $\mathcal{X}Reg$ [31], we can deduce that even for downward closed queries, $Q_1 \preccurlyeq_{2,MSO} Q_2$ does not imply $Q_1 \preccurlyeq_{2,\mathcal{X}Reg} Q_2$. Furthermore, in terms of expressivity, the queries Q_1 and Q_2 used in the proof belong to a small fragment of $\mathcal{X}Reg$ in that they do not use the full expressivity of the Kleene star. This means that when the depth of the domain is not bounded, in general, given any fragment $\mathcal{C}$ of $\mathcal{X}Reg$ and queries $Q_1', Q_2' \in \mathcal{C}$, $Q_1' \preccurlyeq_2 Q_2'$ does not imply $Q_1' \preccurlyeq_{2,\mathcal{C}} Q_2'$ as soon as $\mathcal{C}$ is expressive enough to define Q_1 and Q_2.

Corollary 4. *There exist two downward-closed queries Q_1 and Q_2 given by $\mathcal{X}Reg$ expressions such that $Q_1 \preccurlyeq_2 Q_2$ but $Q_1 \not\preccurlyeq_{2,\mathcal{X}Reg} Q_2$.*

Because the third comparison (like $\preccurlyeq_2$) is essentially independent from any query language, these difficulties due to the expressiveness of $\mathcal{X}Reg$ do not apply when comparing $\mathcal{X}Reg$ queries w.r.t. comparison $\preccurlyeq_3$.

Proposition 14. *The problem of deciding $\preccurlyeq_3$ for interval bounded $\mathcal{X}Reg$ queries can be decided in exponential time.*

Proof. The proof first translates the $\mathcal{X}Reg$ expressions into automata using [31], and proceeds as for Theorem 9: even if the automata that recognize A_{Q_1} and A_{Q_2} have exponential size, the overall complexity remains exponential. □

As usual, this complexity drops to Pspace when the depth of the domain is bounded by the size of the query:

Proposition 15. *The problem of deciding $Q_1 \preccurlyeq_3 Q_2$ for $\mathcal{X}Reg$ queries Q_1 and Q_2 over non-recursive DTD D can be decided in polynomial space.*

Proof. We adapt the proof of Theorem 8. Once more, we use the translation from $\mathcal{X}Reg$ expressions into automata, to build an automaton $\mathcal{A}$ of exponential size with language $L(\mathcal{A}) = \{t \otimes A_{Q_1} \otimes A_{Q_2} \mid t \in L(D)\}$. Actually, we do not build the automaton, because it is of exponential size. But since the depth is bounded, we can simulate its transitions in polynomial space, which also implies we can simulate in polynomial space the transitions of $\mathcal{B}_w$ -where $\mathcal{B}_w$ is defined from $\mathcal{A}$ as in the proof of Theorem 8. The proof proceeds as for Theorem 8. □

	VPA			$\mathcal{X}Reg$		
SAS	non-rec DTD	IB	gen	non-rec DTD	IB	gen
$\preccurlyeq_1$	PTime	PTime	PTime	Pspace-c	Exptime-c	Exptime-c
$\preccurlyeq_2$	Pspace-c [(1)]	Pspace-c [(2)]	undec	Pspace-c	$Memb^{MSO}_{\mathcal{X}Reg}$ Exptime-h	undec
$\preccurlyeq_3$	Pspace-c [(1)]	Exptime Pspace-h	undec	Pspace-c	Exptime-c	undec

[(1)]: When the depth of the DTD is bounded by a *fixed* integer k, this problem becomes polynomial.
[(2)]: When the constant for interval boundedness is a *fixed* integer k, this problem becomes polynomial.

8. Conclusions and future work

Summary. In this paper, we have first studied the problem of rewriting queries with views, when the classes used to defined queries and views are $\mathcal{X}Reg$ and MSO. In a second part, we have defined different manner to compare views (i.e. on queries), with a security point of view. We suggest three comparisons, the first one being essentially containment, while the second and third one respectively decide if a view can be rewritten in terms of another, and if a view determines another (when we do not consider the identifiers). We provide a systematic study of the decidability and complexity for the three comparisons when the depth of the xml documents is bounded, when the document may have an arbitrary depth but the query defining the policies are restricted to guarantee the interval-boundedness property, and in the general setting without restriction on queries and document.

Related work. The closure under query rewriting has been investigated for several xpath query languages. Benedikt and Fundulaki [2] define *subtree queries* and study their closure under composition. A subtree query can be seen as a downward-closed view, with the additional requirement that leaves of the view trees must be leaves also in the original tree. The authors study for which fragment of xpath (with vertical axes) the subtree queries are closed under rewriting. Vercammen et al. [34] study the closure under composition of xpath. Their setting is similar to ours, but the fragments of xpath they consider are different, as they do not allow the transitive closure operator but allow path intersection and path complementation operators. Some of their fragments are not closed under composition: essentially those that exclude path complementation but including recursive axes, or sibling axes, or union. The remaining fragments studied in the paper are closed under composition. What is more, the time complexity for the rewriting are similar to ours. The approach adopted by Vercammen et al. also relies on the rewriting of the base axes, but exploits path intersection instead of the transitive closure operator. Fan et al. [12, 14] show that the downward fragment of xpath is closed under rewriting for non-recursive views, whereas downward regular xpath is closed under query rewriting for arbitrary views (recursive or not). The complexity for rewriting downward regular xpath, though, is exponential. Fan et al. therefore devise an approach, based on alternating automata, to escape the exponential lower bound on rewriting for their fragment of regular xpath.

Our first comparison corresponds to the inclusion of queries, so the results are deduced immediately from the literature. The second comparison was shown to be equivalent to the problem of rewriting some query in terms of another, which means it is related to the rewriting problems studied, among many others, in [7, 8] for regular expressions and regular path queries, in [10] for tree patterns, or mentioned in [25] for logical queries on graphs. It is also close to the notion of query rewriting as discussed above.

The third is related to the problems known in the literature as the *determinacy* problem [25] or losslessness [8] for different query formalisms. While

the results are negative in the most general setting, leading to undecidability of those comparisons, natural restrictions like interval boundedness give interesting results: namely, using the terminology of [25], Lemma 7 states that *MSO* is *complete* for *MSO*-to IB-*MSO* rewriting, which means that whenever a view Q_V given by an IB-*MSO* query provides enough information to answer another *MSO* query Q (posed on the source document), then we can rewrite Q in terms of Q_V using another *MSO* query Q'. In other words, given any interval-bounded view Q_V and query Q, either there exists a *MSO* query Q' such that $\mathsf{Rewrite}(Q', Q_V) = Q$ or Q_V does not provide enough information to answer query Q and the expressiveness of *MSO* is not involved in the impossibility to rewrite Q in terms of Q_V. In contrast, $\mathcal{X}Reg$ queries do not have this property, even with the interval-boundedness restriction. We prove that deciding the second comparison for interval-bounded security access specification can be reduced to testing whether a *MSO* query can be expressed via an equivalent $\mathcal{X}Reg$ query, a problem whose decidability is still open. Moreover, Proposition 13 exhibits a polynomial time reduction from this open problem to the comparison of interval-bounded $\mathcal{X}Reg$ access specification. We obtain PSPACE-hardness lower bounds, even when interval-boundedness restrictions are enforced. However, reasonable restrictions on the constant k for interval-boundedness provide tractable cases with polynomial algorithms.

Another approach to guarantee privacy is mentioned in [21], that requires the administrator to define the information he considers secret using a boolean first order query $\mathcal{Q}$. The secret is considered to be revealed by the view Q_V if there exists some document t such that $\mathit{Certain}_{Q_V}(\mathcal{Q}; \mathcal{V}\mathit{iew}(Q_V, t)) = \mathsf{true}$, i.e. if there is a view document from which we can guess that $\mathcal{Q}$ holds. Given a boolean query $\mathcal{Q}$ and a view Q_V, one has to check whether there exists a such document t that reveals $\mathcal{Q}$. [21] proves this problem is undecidable in general, and decidable for downward closed views, in polynomial time if the queries are given by an automaton, and exponential time if they are given by a ConditionalXPath formula. Since the proof only requires the regularity of $\mathcal{V}\mathit{iew}(Q_V, D)$, their proof of decidability can be extended to interval-bounded views. This approach allows more precise verification of the policy, and could replace the comparison of policies, except that this flexibility comes at the price of a weaker guarantee. Actually, this definition of secret might be vulnerable to probabilistic attacks in case of a priori knowledge or assumptions on the source document, while comparison $\preccurlyeq_2$ is so restrictive that it really provides a strong guarantee that view A_1 discloses less information than view A_2 as soon as $A_1 \preccurlyeq_2 A_2$.

Bohannon et al. [4] study related notions though in a very different framework. They consider a function that maps each document satisfying one DTD D_1 to a document satisfying another DTD D_2. Such a function σ is *invertible* if the original document can be recovered from the target document. Similarly, the function σ is *query preserving with respect to a query language* L if there is a computable function $F : L \to L$ such that for any $Q \in L$ and any document t satisfying D_1, $Q(t) = F(Q)(\sigma(t))$. In short, σ is query preserving w.r.t. L if every query from L can be rewritten as the composition of another query from

L with σ. The paper considers schema mappings, whereas we consider views. What is more, we distinguish two settings depending on whether identifiers are taken into account or not, a distinction that is absent from [4]. Their definition of invertibility may be considered under both settings. We observe that when considering identifiers, $Q_1 \preccurlyeq_2 Q_2$ if and only if Q_2 is query-preserving w.r.t. $\mathit{Public}(Q_1)$. Also, $\mathrm{Id} \preccurlyeq_3 V$ if and only if V is invertible in the sense of [4], where Id is the identity query, i.e., the view that hides no node. For this, we must consider invertibility without identifiers in our model: if we assume each node has a (unique and arbitrary) identifier, every query V that deletes an unbounded number of nodes would not be invertible due to the impossibility to recover the identifiers of the hidden nodes.

Future work and discussion. MSO views could also be defined differently, for instance by using automata with selecting states, or automata over alphabet $\Sigma \times \{\mathbb{0}, \mathbb{1}\}$ such that the nodes selected in a tree $t \in T_\Sigma$ over alphabet Σ are: $\{n \in N_t \mid \exists t' \in T_{\Sigma\times\{\mathbb{0},\mathbb{1}\}}. \Pi_\Sigma(t') = t, \lambda_{t'}(n) \in \Sigma \times \{\mathbb{1}\}\}$, without requiring maximality of the languages. However inclusion of queries represented by such automata is already PSPACE-complete over non-recursive DTDs.

All results can be generalized to the setting of views and queries that not only select nodes, but also rename some of them: instead of recognizing (maximal) languages over alphabet $\Sigma \times \{\mathbb{0}, \mathbb{1}\}$, the automata could have been defined over alphabet $\Sigma \times \Sigma$, a node labeled with (a, b) representing the relabeling of a into b. We could then adapt the comparisons of security access specifications to this setting, which would lead to similar results using mostly the same constructions. However, dealing with more general cases, where insertions, copying or restructuring are allowed e.g., would require other techniques.

Our model of security view is based on nodes authorizations, whereas administrator could want to express also relationships authorizations [15]. This work could be developed into several directions, like considering views defined by other kinds of transducers allowing to modify the structure of the document instead of monadic queries. Our framework could at least be extended to deal with n-ary queries from the user, while keeping the view defined by a monadic one. One may also study other restrictions that would allow polynomial algorithms.

Regarding the rewriting problem studied in the first part of this paper, it would be interesting to study how the knowledge of the domain -in the case of annotated DTD- could be incorporated to optimize the quadratic query rewriting.

Acknowledgements: The authors thank Iovka Boneva and Yves André for fruitful discussions on access control and security views.

References

[1] Rajeev Alur and P. Madhusudan. Visibly pushdown languages. In *STOC*, pages 202–211, 2004.

[2] M. Benedikt and I. Fundulaki. XML subtree queries: Specification and composition. In *International Symposium on Database Programming Languages (DBPL)*, pages 138–153, 2005.

[3] Patrick Blackburn, Maarten de Rijke, and Yde Venema. *Modal logic*. Cambridge University Press, New York, NY, USA, 2001.

[4] Philip Bohannon, Wenfei Fan, Michael Flaster, and P. P. S. Narayan. Information preserving XML Schema embedding. In *VLDB*, pages 85–96, 2005.

[5] A. R. Büchi. Weak second-order arithmetic and finite automata. *Z. Math. Logik Grundlagen Math*, 6:66–92, 1960.

[6] D. Calvanese, G. De Giacomo, M. Lenzerini, and M.Y. Vardi. An automata-theoretic approach to regular XPath. In *DBPL*, pages 18–35, 2009.

[7] Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, and Moshe Y. Vardi. Rewriting of regular expressions and regular path queries. *J. Comput. Syst. Sci.*, 64(3):443–465, 2002.

[8] Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, and Moshe Y. Vardi. View-based query processing: On the relationship between rewriting, answering and losslessness. *Theor. Comput. Sci.*, 371(3):169–182, 2007.

[9] Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, and Moshe Y. Vardi. An automata-theoretic approach to regular xpath. In Philippa Gardner and Floris Geerts, editors, *DBPL*, volume 5708 of *Lecture Notes in Computer Science*, pages 18–35. Springer, 2009.

[10] Bogdan Cautis, Alin Deutsch, Nicola Onose, and Vasilis Vassalos. Efficient rewriting of xpath queries using query set specifications. *PVLDB*, 2(1):301–312, 2009.

[11] Hubert Comon, Max Dauchet, Rémi Gilleron, Christof Löding, Florent Jacquemard, Denis Lugiez, Sophie Tison, and Marc Tommasi. Available online since 1997: `http://tata.gforge.inria.fr`, October 2007.

[12] W. Fan, C.-Y. Chan, and M. N. Garofalakis. Secure XML querying with security views. In *ACM SIGMOD International Conference on Management of Data*, pages 587–598, 2004.

[13] W. Fan, F. Geerts, X. Jia, and A. Kementsietsidis. SMOQE: A system for providing secure access to XML. In *International Conference on Very Large Data Bases (VLDB)*, pages 1227–1230. ACM, 2006.

[14] W. Fan, F. Geerts, X. Jia, and A. Kementsietsidis. Rewriting regular XPath queries on XML views. In *International Conference on Data Engineering (ICDE)*, pages 666–675, 2007.

[15] Béatrice Finance, Saïda Medjdoub, and Philippe Pucheral. The case for access control on xml relationships. In *Proceedings of the 14th ACM international conference on Information and knowledge management*, CIKM '05, pages 107–114, New York, NY, USA, 2005. ACM.

[16] Olivier Gauwin. *StreamingTree automata and XPath*. PhD thesis, University Lille I, September 2009. Available online at http://hal.inria.fr/tel-00421911/en.

[17] E.M. Gurari and O.H Ibarra. The complexity of decision problems for finite-turn multicounter machines. *J. Comput. Syst. Sci.*, 22(2):220–229, 1981.

[18] E.M. Gurari and O.H Ibarra. A note on finitely-valued and finitely ambiguous transducers. *Mathematical Systems Theory*, 16(1):61–66, 1983.

[19] G. Kuper, F. Massacci, and N. Rassadko. Generalized XML security views. In *SACMAT '05: Proceedings of the tenth ACM Symposium on Access Control Models and Technologies*, pages 77–84. ACM, 2005.

[20] L. Libkin and C. Sirangelo. Reasoning about XML with temporal logics and automata. In *International Conference on Logic for Programming, Artificial Intelligence, and Reasoning (LPAR)*, volume 5330, pages 97–112. Springer LNAI, 2008.

[21] L. Libkin and C. Sirangelo. Reasoning about XML with temporal logics and automata. *J. Applied Logic*, 8(2):210–232, 2010.

[22] Markus Lohrey. Compressed membership problems for regular expressions and hierarchical automata. *Int. J. Found. Comput. Sci.*, 21(5):817–841, 2010.

[23] M. Marx. XPath with conditional axis relations. In *International Conference on Extending Database Technology (EDBT)*, pages 477–494, 2004.

[24] Makoto Murata, Akihiko Tozawa, Michiharu Kudo, and Satoshi Hada. XML access control using static analysis. *ACM Trans. Inf. Syst. Secur.*, 9(3):292–324, 2006.

[25] Alan Nash, Luc Segoufin, and Victor Vianu. Views and queries: Determinacy and rewriting. *ACM Trans. Database Syst.*, 35(3), 2010.

[26] Wojciech Plandowski. Testing equivalence of morphisms on context-free languages. In *ESA*, pages 460–470, 1994.

[27] N. Rassadko. Policy classes and query rewriting algorithm for XML security views. In *20th Annual IFIP WG 11.3 Working Conference on Data and Applications Security (DBSec)*, volume 4127 of *Lecture Notes in Computer Science*, pages 104–118. Springer, 2006.

[28] N. Rassadko. Query rewriting algorithm evaluation for XML security views. In *Secure Data Management (VLDB Workshop)*, volume 4721 of *Lecture Notes in Computer Science*, pages 64–80. Springer, 2007.

[29] A. Stoica and C. Farkas. Secure XML views. In *IFIP WG 11.3 Sixteenth International Conference on Data and Applications Security*, volume 256 of *Research Directions in Data and Applications Security*, pages 133–146. Kluwer, 2002.

[30] T. G. Szymanski and J. H. Williams. Non-canonical parsing. In *14th Annual Symposium on Foundations of Computer Science*, pages 122–129. IEEE, 1973.

[31] B. ten Cate and L. Segoufin. XPath, transitive closure logic, and nested tree walking automata. In *ACM Symposium on Principles of Database Systems (PODS)*, pages 251–260, 2008.

[32] Balder ten Cate. The expressivity of XPath with transitive closure. In *PODS*, pages 328–337, 2006.

[33] J. W. Thatcher and Wright J. B. Generalized finite automata with an application to a decision problem of second-order logic. *Mathematical System Theory*, 2:57–82, 1968.

[34] R. Vercammen, J. Hidders, and J. Paredaens. Query translation for XPath-based security views. In *EDBT Workshops*, volume 4254 of *Lecture Notes in Computer Science*, pages 250–263. Springer, 2006.

9. Appendix

In order to prove Theorem 9, we could first think of adapting immediately the proof of Lemma 8 in order to use Proposition 3. Let (t, t') be a pair of trees of minimal size such that $\mathcal{V}iew\,(Q_2, t) \sim \mathcal{V}iew\,(Q_2, t')$ but $\mathcal{V}iew\,(Q_1, t) \not\sim \mathcal{V}iew\,(Q_1, t')$. Let ϕ denote an isomorphism between $\mathcal{V}iew\,(Q_2, t)$ and $\mathcal{V}iew\,(Q_2, t')$.

Suppose there are three nodes $n_t^{\uparrow}, n_t^{\circ}, n_t^{\downarrow}$ in $Q_2(t)$, and three nodes $n_{t'}^{\uparrow}, n_{t'}^{\circ}, n_{t'}^{\downarrow}$ such that $n_t^{\uparrow}$ is an ancestor of n_t°, n_t° an ancestor of $n_t^{\downarrow}$, $\phi(n_t^{\uparrow}) = n_{t'}^{\uparrow}$, $\phi(n_t^{\circ}) = n_{t'}^{\circ}$, $\phi(n_t^{\downarrow}) = n_{t'}^{\downarrow}$, $\rho_t(n_t^{\uparrow}) = \rho_t(n_t^{\circ}) = \rho_t(n_t^{\downarrow})$ and $\rho_{t'}(n_{t'}^{\uparrow}) = \rho_{t'}(n_{t'}^{\circ}) = \rho_{t'}(n_{t'}^{\downarrow})$, where ρ_t, ρ'_t are defined similarly to ρ in Lemma 8. Replacing the subtrees below $n_t^{\uparrow}$ (resp. $n_{t'}^{\uparrow}$) with the subtree below n_t° (resp. $n_{t'}^{\circ}$), we preserve isomorphic views for Q_2. However, the views for Q_1 may become isomorphic. One could think that at least one of the combinations for the pumping would make sure the views for Q_1 remain non isomorphic. It so happens that this is not true, as illustrated in Figure 8. In this figure, Q_2 selects all the nodes labeled with d, plus the root, and Q_1 selects all the nodes with label different from d. Clearly, $\mathcal{V}iew\,(Q_2, t) \sim \mathcal{V}iew\,(Q_2, t')$ and $\mathcal{V}iew\,(Q_1, t) \not\sim \mathcal{V}iew\,(Q_1, t')$. We can build the

automata for Q_2 and Q_1 such that ρ_t (resp. $\rho_{t'}$) has the same value on all nodes labeled d in t (resp. t'). However, whatever combination is chosen for the pumping, the views for Q_1 become isomorphic after we replace the subtrees. For instance, if we replace the subtree below $n^{\uparrow}$ with the subtree below n° in both trees, the views obtained for Q_1 are both isomorphic to $r(c(a))$, and if we replace the subtree below n° with the subtree below $n^{\downarrow}$ in both trees, the views obtained for Q_1 are both isomorphic to $r(a,b,a)$. So, there is no trivial adaptation from the proof of Lemma 8, and it is not clear how to adapt this pumping lemma. Therefore, we developped a new method, based on alignment of trees, which we discuss hereunder.

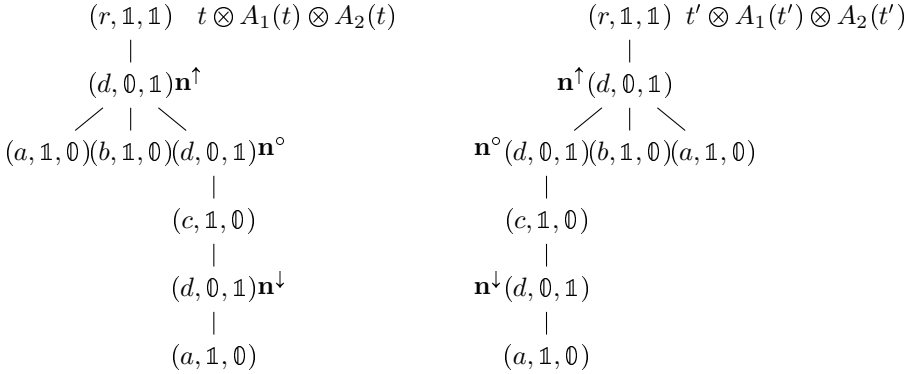


Figure 8: The pumping of Lemma 8 does not work for $\preccurlyeq_3$

Caveat: In this whole proof, we consider trees as terms, i.e., we do not consider identifiers. Two trees will be considered equal iff they are isomorphic. We also define an hedge as a sequence of trees.

We define alphabet Σ_{align} as $\Sigma_{align} = \Sigma^2 \cup (\Sigma \times \{\mathbb{0}\}) \cup (\{\mathbb{0}\} \times \Sigma)$. Given two trees t_1, t_2 over Σ_{align}, we denote by $t_1 \boxtimes t_2$ the *square* of t_1 and t_2, i.e., the tree defined by the recursive algorithm hereunder. Similarly, given two (IB)alignment languages L_1 and L_2, we define $L_1 \boxtimes L_2$ as $\{t_1 \boxtimes t_2 \mid t_1 \in L_1, t_2 \in L_2\}$.

A recursive definition for $t_1 \boxtimes t_2$. We define more generally operation $\boxtimes$ as a binary operation on hedges.

We note hedges as follow: $\cdot$ represents the concatenation of hedges, $f[h]$ represents the tree with root f and, such that, if $h = t_1 \cdot t_2 \cdot \ldots \cdot t_k$, then $f[h]$ is the tree $f(t_1, t_2, \ldots, t_k)$. Hence, $f[a[b \cdot c] \cdot d] \cdot g$ represents the hedge $f(a(b,c),d) \cdot g$. To avoid confusion, we note pairs/triples of symbols (i.e. tags over product alphabets like $\Sigma \times \Sigma$) between "$\langle$", "$\rangle$" instead of usual parenthesis. We fix the following priorities for operations: insertion $[\,]$ of an hedge under a node has highest priority, next comes the concatenation $\cdot$ of two hedges, and $\boxtimes$ has the lowest priority. The rules are as follows: for every letters $a, b \in \Sigma$, $\alpha_1, \alpha_2 \in \Sigma \cup \{\mathbb{0}\}$, every hedges h_1, h_2, w_1, w_2,

1. $(\langle b, \alpha_1\rangle[h_1]\cdot w_1)\boxtimes(\langle b, \alpha_2\rangle[h_2]\cdot w_2) = \langle b, \alpha_1, \alpha_2\rangle[h_1\boxtimes h_2]\cdot(w_1\boxtimes w_2)$
2. $(\langle \mathbb{0}, a\rangle[h_1]\cdot w_1)\boxtimes h'$ is defined as:
   $\begin{cases} \langle \mathbb{0}, a, \mathbb{0}\rangle[\mathcal{T}(h_1)]\cdot(w_1\boxtimes h') & \text{if } h_1 \text{ is a hedge over } \{\mathbb{0}\}\times\Sigma \\ (\langle \mathbb{0}, a, \text{op}\rangle\cdot h_1\cdot\langle \mathbb{0}, a, \text{cl}\rangle\cdot w_1)\boxtimes h' & \text{otherwise} \end{cases}$
   where $\mathcal{T}$ is defined by $\mathcal{T}(\langle \mathbb{0}, c\rangle[h]\cdot w) = \langle \mathbb{0}, c, \mathbb{0}\rangle[\mathcal{T}(h)]\cdot\mathcal{T}(w)$ and the image by $\mathcal{T}$ of the empty word(neutral element of the monoid) is the empty word.
3. $(\langle \mathbb{0}, a, \text{op}\rangle\cdot w_1)\boxtimes h' = \langle \mathbb{0}, a, \mathbb{0}, \text{op}\rangle\cdot(w_1\boxtimes h')$ [2]. Symmetrically, for closing tags, $(\langle \mathbb{0}, a, \text{cl}\rangle\cdot w_1)\boxtimes h' = \langle \mathbb{0}, a, \mathbb{0}, \text{cl}\rangle\cdot(w_1\boxtimes h')$.
4. for the right operand, we add the three symmetrical rules $h\boxtimes\langle \mathbb{0}, a\rangle[h_2]\cdot w_2$, $h\boxtimes(\langle \mathbb{0}, a, \text{op}\rangle\cdot w_2)$, and $h\boxtimes(\langle \mathbb{0}, a, \text{cl}\rangle\cdot w_2)$. The second rule, for instance, is $h'\boxtimes(\langle \mathbb{0}, a, \text{op}\rangle\cdot w_1) = \langle \mathbb{0}, \mathbb{0}, a, \text{op}\rangle\cdot(w_1\boxtimes h')$. However, in order to get at most one result, we fix that rules 2 and 3 have higher priority than their 'right' counterpart. Thus, the 'right' rules can be applied only if no left one can.

This definition is extended to languages by $L_1\boxtimes L_2 = \{t_1\boxtimes t_2 \mid t_1 \in L_1, t_2 \in L_2\}$.

Example 7. In figure 9, we represent two alignment trees and their square.

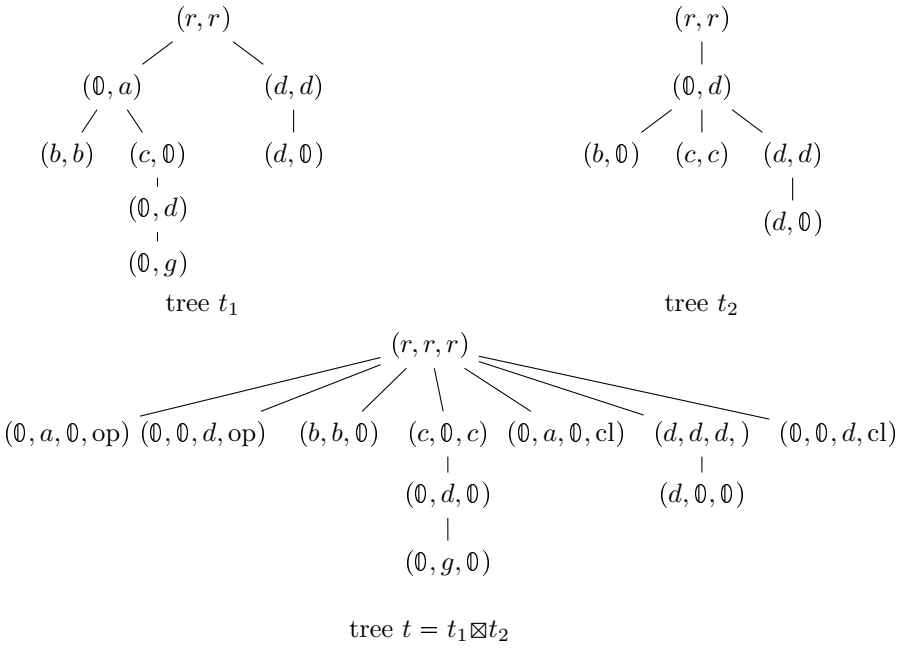


Figure 9: Two alignment trees and their square

[2] the construction fails if a node of the form $\langle \mathbb{0}, a, \text{op}\rangle$ has a child

We define two morphisms ϕ_1, ϕ_2 on linearization(and, by abuse, on trees):

- $\forall \eta \in \{\mathrm{op}, \mathrm{cl}\}, \forall a \in \Sigma, \forall \alpha_1, \alpha_2 \in \Sigma \cup \{\mathbb{0}\}, \forall i \in \{1,2\}, \phi_i(\eta, a, \alpha_1, \alpha_2) = (\eta, a, \alpha_i)$ if $\alpha_i \in \Sigma, \epsilon$[3] otherwise .
- $\forall \eta, \eta' \in \{\mathrm{op}, \mathrm{cl}\}, \forall a \in \Sigma$, $\phi_1(\mathrm{op}, \mathbb{0}, a, \mathbb{0}, \eta') = (\eta', \mathbb{0}, a)$, $\phi_1(\mathrm{cl}, \mathbb{0}, a, \mathbb{0}, \eta') = \epsilon^1$, and $\phi_1(\eta, \mathbb{0}, \mathbb{0}, a, \eta') = \epsilon^1$. Similarly, $\phi_2(\mathrm{op}, \mathbb{0}, \mathbb{0}, a, \eta') = (\eta', \mathbb{0}, a)$, $\phi_2(\mathrm{cl}, \mathbb{0}, \mathbb{0}, a, \eta') = \epsilon^1$, and $\phi_2(\eta, \mathbb{0}, a, \mathbb{0}, \eta') = \epsilon^1$.
- $\forall \eta \in \{\mathrm{op}, \mathrm{cl}\}, \forall a \in \Sigma$, $\phi_1(\eta, \mathbb{0}, a, \mathbb{0}) = (\eta, \mathbb{0}, a)$, and $\phi_1(\eta, \mathbb{0}, \mathbb{0}, a) = \epsilon^1$.
  Similarly, $\phi_2(\eta, \mathbb{0}, \mathbb{0}, a) = (\eta, \mathbb{0}, a)$, and $\phi_2(\eta, \mathbb{0}, a, \mathbb{0}) = \epsilon^1$.

We also denote by π_1 the following morphism on linearization(and, by abuse, on trees). For all $a \in \Sigma$, all $\beta \in \Sigma \cup \{\mathbb{0}\}$, $\pi_1(a, \beta) = a$ and $\pi_1(\mathbb{0}, \beta) = \epsilon^1$. Intuitively, it represents the projection on first component, where nodes with label $\mathbb{0}$ are deleted (and a node whose father has been deleted is adopted by its closest "non-deleted" ancestor).

Proposition 16. *For every two trees t_1 and t_2 over Σ_{align}, $t_1 \boxtimes t_2$ exists iff $\pi_1(t_1) = \pi_1(t_2)$, in which case it is a unique tree, $t_1 = \phi_1(t_1 \boxtimes t_2)$ and $t_2 = \phi_2(t_1 \boxtimes t_2)$.*

Proof. $t_1 \boxtimes t_2$ exists iff $\pi_1(t_1) = \pi_1(t_2)$(recall that equality means isomorphism) because rule 1 is the only rule that allows a tag in Σ on the first component, and this rule requires that the same letter b occurs at the same position in $\pi_1(t_1)$ and $\pi_1(t_2)$. Clearly, this is also a sufficient condition for the existence of $t_1 \boxtimes t_2$. The priority rules make the algorithm deterministic: only one rule can be applied at any time, which guarantees the uniqueness. As for $t_1 = \phi_1(t_1 \boxtimes t_2)$ and $t_2 = \phi_2(t_1 \boxtimes t_2)$, it can be proved by induction, analysing each of the rules.

We denote by $V_{2\to 1}$ the function that maps each tree t over Σ to the tree t' over Σ_{align} defined by $N_{t'} = \{n \in N_t \mid A_2(n) = \mathbb{1} \vee A_1(n) = \mathbb{1}\}$, $descendants_{t'} = descendants_t \cap N_{t'}^2$, $following_{t'} = following_t \cap N_{t'}^2$, [4] and $\lambda_{t'}(n) = (\alpha, \beta)$ where $\alpha = \mathbb{0}$ if $A_2(n) = \mathbb{0}$, and $\alpha = \mathbb{1}$ otherwise, while $\beta = \mathbb{0}$ if $A_1(n) = \mathbb{0}$, and $\beta = \mathbb{1}$ otherwise. This definition is extended to languages by $V_{2\to 1}(L) = \bigcup_{t \in L} V_{2\to 1}(t)$.

Proposition 17. *Given two k-interval bounded root preserving queries Q_1 and Q_2 with $\mathrm{dom}(Q_1) = \mathrm{dom}(Q_2) = D$, there is a polynomial p_0 such that one can compute an automaton $\mathcal{B}$ that recognizes $V_{2\to 1}(D)$ in time $(|A_{Q_1}| + |A_{Q_2}|)^{p_0(k)}$*

Proof. We can first build an automaton $\mathcal{B}_0$ that recognizes $L(\mathcal{B}_0) = \{t \otimes A_{Q_1} \otimes A_{Q_2} \mid t \in L(D)\}$ in polynomial time. $\mathcal{B}$ is built from $\mathcal{B}_0$ by simulating the transitions through the nodes labeled $(a, \mathbb{0}, \mathbb{0})$ for all $a \in \Sigma$. This can be achieved using tricks similar to the construction for Lemma 7, working in time exponential in k.

[3] the neutral element of the free monoid

[4] for the time being trees are defined using children , not descendants, and next, not following so we need to adapt....

Remark 2. Due to the k-interval boundedness of A_2, $V_{2\to1}(D)$ presents the following property: for every t in $V_{2\to1}(D)$, for every nodes $n_1, n_2, \ldots, n_{k+1} \in N_t$, with $(n_1, n_2) \in child_t$, $(n_2, n_3) \in child_t$ ..., and $(n_k, n_{k+1}) \in child_t$, if $\lambda_t(n_1) \in \{\mathbb{0}\} \times \Sigma$, $\lambda_t(n_2) \in \{\mathbb{0}\} \times \Sigma$, ...and $\lambda_t(n_{k+1}) \in \{\mathbb{0}\} \times \Sigma$, then for every descendant n' of n_{k+1}, $\lambda_t(n') \in \{\mathbb{0}\} \times \Sigma$.

Proposition 18. *Given two k-interval bounded root preserving queries Q_1 and Q_2 with $\mathrm{dom}(Q_1) = \mathrm{dom}(Q_2) = D$, there is a polynomial p such that one can compute an automaton $\mathcal{B}_{align}$ that recognizes $V_{2\to1}(D) \boxtimes V_{2\to1}(D)$ in time $(|A_{Q_1}| + |A_{Q_2}|)^{p(k)}$*

Proof. Actually this holds not only for $V_{2\to1}$, but also for every language presenting the property in Remark 2. Let $\mathcal{B} = (\Sigma_{align}, Q, \Gamma, I, F, R)$ be the automaton recognizing $V_{2\to1}(D)$ as in Proposition 17. We define automaton $\mathcal{B}_{align}$ as $(\Sigma', Q', \Gamma', I', F', R')$ where:

- $\Sigma' = \Sigma_1 \cup \Sigma_2 \cup \Sigma_3$ with

$$\begin{aligned}\Sigma_1 &= \Sigma \times (\Sigma \cup \{\mathbb{0}\}) \times (\Sigma \cup \{\mathbb{0}\}),\\ \Sigma_2 &= (\{\mathbb{0}\} \times \{\mathbb{0}\} \times \Sigma) \cup (\{\mathbb{0}\} \times \Sigma \times \{\mathbb{0}\}),\\ \Sigma_3 &= (\{\mathbb{0}\} \times \{\mathbb{0}\} \times \Sigma \times \{\mathrm{op}, \mathrm{cl}\}) \cup (\{\mathbb{0}\} \times \Sigma \times \{\mathbb{0}\} \times \{\mathrm{op}, \mathrm{cl}\}).\end{aligned}$$

- $Q' = Q_{13} \cup Q_2$ where $Q_{13} = Q \times Q \times \Gamma^{\leqslant k} \times \Gamma^{\leqslant k} \times \{\top, \bot\} \times \{C_l, C_r\}$ and $Q_2 = (Q \times \{\sharp\}) \cup (\{\sharp\} \times Q)$
- $\Gamma' = \Gamma_{13} \cup \Gamma_2$ where $\Gamma_{13} = \Gamma \times \Gamma \times \Gamma^{\leqslant k} \times \Gamma^{\leqslant k} \times \{\top, \bot\}$ and $\Gamma_2 = \Gamma \cup (\Gamma \times \Gamma^{\leqslant k} \times \Gamma^{\leqslant k} \times \{\top, \bot\} \times Q)$
- $I' = \{(q_l, q_r, \varepsilon, \varepsilon, \bot) \mid q_l, q_r \in I\}$
- $F' = \{(q_l, q_r, \varepsilon, \varepsilon, \bot) \mid q_l, q_r \in F\}$
- the rules in R' are defined as follows: for all $q_l, q_r, q'_l, q'_r \in Q$, all $\gamma_l, \gamma_r \in \Gamma$, all $u_l, u_r \in \Gamma^{\leqslant k}$, all $\eta \in \{\bot, \top\}$, all $C \in \{C_l, C_r\}$, all $\alpha_1, \alpha_2 \in \Sigma \cup \{\mathbb{0}\}$, all $\theta \in \{\mathrm{op}, \mathrm{cl}\}$ and all $b \in \Sigma$;
  - $(q_l, q_r, u_l, u_r, \eta, C) \xrightarrow{(\mathrm{op},(b,\alpha_1,\alpha_2)):(\gamma_l,\gamma_r,u_l,u_r,\bot,C_l)} (q'_l, q'_r, \varepsilon, \varepsilon, \bot)$ is in R' if there are rules $q_l \xrightarrow{(\mathrm{op},(b,\alpha_1)):\gamma_l} q'_l$ and $q_r \xrightarrow{(\mathrm{op},(b,\alpha_2)):\gamma_r} q'_r$ in R.
  - $(q_l, q_r, \varepsilon, \varepsilon, \bot, C) \xrightarrow{(\mathrm{cl},(b,\alpha_1,\alpha_2)):(\gamma_l,\gamma_r,u_l,u_r,\bot,C_l)} (q'_l, q'_r, u_l, u_r, \bot)$ is in R' if there are rules $q_l \xrightarrow{(\mathrm{cl},(b,\alpha_1)):\gamma_l} q'_l$ and $q_r \xrightarrow{(\mathrm{cl},(b,\alpha_2)):\gamma_r} q'_r$ in R.
  - $(q_l, q_r, u_l, u_r, \eta, C_l) \xrightarrow{(\mathrm{op},(\mathbb{0},b,\mathbb{0},\mathrm{op}))(\mathrm{cl},(\mathbb{0},b,\mathbb{0},\mathrm{op}))} (q'_l, q_r, u_l \cdot \gamma_l, u_r, \top, C_l)$ is in R' if there is a rule $q_l \xrightarrow{(\mathrm{op},(\mathbb{0},b)):\gamma_l} q'_l$ in R and $u_l \in \Gamma^{\leqslant k-1}$.
    We use a transition that does not modify the stack and reads two symbols at a time for the sake of clarity. Actually, this does not

strictly follow the syntax of VPA transitions. However, it is straightforward to introduce a few new states to simulate this behaviour with two transitions, the first transition pushing a symbol into the stack which is immediately removed by the second one.

- $(q_l, q_r, u_l \cdot \gamma_l, u_r, \bot, C_l) \xrightarrow{(\mathrm{op},(\mathbb{0},b,\mathbb{0},\mathrm{cl}))(\mathrm{cl},(\mathbb{0},b,\mathbb{0},\mathrm{cl}))} (q'_l, q_r, u_l, u_r, \bot, C_l)$ is in R' if there is a rule $q_l \xrightarrow{(\mathrm{cl},(\mathbb{0},b)):\gamma_l} q'_l$ in R and $u_l \in \Gamma^{\leqslant k-1}$.
- The rules for $(\mathbb{0},\mathbb{0},b,op)$ and $(\mathbb{0},\mathbb{0},b,\mathrm{cl})$ are symmetric, except for the C_l, C_r constraints that need to be adapted, yielding rules $(q_l, q_r, u_l, u_r, \eta, C) \xrightarrow{(\mathrm{op},(\mathbb{0},\mathbb{0},b,\mathrm{op}))(\mathrm{cl},(\mathbb{0},\mathbb{0},b,\mathrm{op}))} (q'_l, q_r, u_l \cdot \gamma_l, u_r, \top, C_r)$ and $(q_l, q_r, u_l \cdot \gamma_l, u_r, \bot, C) \xrightarrow{(\mathrm{op},(\mathbb{0},\mathbb{0},b,\mathrm{cl}))(\mathrm{cl},(\mathbb{0},\mathbb{0},b,\mathrm{cl}))} (q'_l, q_r, u_l, u_r, \bot, C_r)$.
- $(q_l, q_r, u_l, u_r, \eta, C_l) \xrightarrow{(\mathrm{op},(\mathbb{0},b,\mathbb{0})):(\gamma_l,u_l,u_r,\eta,q_r)} (q'_l, \sharp)$ is in R' if there is a rule $q_l \xrightarrow{(\mathrm{op},(\mathbb{0},b)):\gamma_l} q'_l$ in R.
- $(q_l, \sharp) \xrightarrow{(\mathrm{cl},(\mathbb{0},b,\mathbb{0})):(\gamma_l,u_l,u_r,\eta,q_r)} (q'_l, q_r, u_l, u_r, \eta, C_l)$ is in R' if there is a rule $q_l \xrightarrow{(\mathrm{cl},(\mathbb{0},b)):\gamma_l} q'_l$ in R.
- $(q_l, \sharp) \xrightarrow{(\theta,(\mathbb{0},b,\mathbb{0})):\gamma_l} (q'_l, \sharp)$ is in R' if rule $q_l \xrightarrow{(\theta,(\mathbb{0},b)):\gamma_l} q'_l$ is in R.
- Rules for $(\mathbb{0},\mathbb{0},b)$ are symmetric, using states in $\{\sharp\} \times Q$ instead of $Q \times \{\sharp\}$, and replacing C_l with C_r.

Basically, we build a product automaton, and the difficulty stems from the synchronization of the stacks. The stacks are synchronized on transitions that read a letter in Σ_1. The state and stack use words u_l, u_r to simulate the runs on letters in Σ_2. The property in Remark 2 allows to bound by k the required size for u_l and u_r. To guarantee the uniqueness property, the definition of the $\boxtimes$ operation demands that we read a letter in Σ_1 between an opening tag $(\mathbb{0},b,\mathbb{0},op)$ and the corresponding closing tag $(\mathbb{0},b,\mathbb{0},\mathrm{cl})$. We use $\top$ to remember this information that one has to read a letter in Σ_1 before reading the next closing tag in Σ_3. $\bot$ is used whenever there is no such constraint. Also for uniqueness, rules 2 and 3 have higher priority than their 'right' counterpart. So, no node with label in $(\{\mathbb{0}\} \times \{\mathbb{0}\} \times \Sigma)$ or $(\{\mathbb{0}\} \times \{\mathbb{0}\} \times \Sigma \times \{\mathrm{op},\mathrm{cl}\})$ can be the left sibling of a node with label in $(\{\mathbb{0}\} \times \Sigma \times \{\mathbb{0}\})$ or $\cup(\{\mathbb{0}\} \times \Sigma \times \{\mathbb{0}\} \times \{\mathrm{op},\mathrm{cl}\})$. We use C_r to remember this information: a tag C_r in the state forbids transition labeled by $(\{\mathbb{0}\} \times \Sigma \times \{\mathbb{0}\})$ or $\cup(\{\mathbb{0}\} \times \Sigma \times \{\mathbb{0}\} \times \{\mathrm{op},\mathrm{cl}\})$. Last, but not least, all descendants of a node of the form $(\mathbb{0},\mathbb{0},b)$ in $t_1 \boxtimes t_2$ have label in $\{\mathbb{0}\} \times \{\mathbb{0}\} \times \Sigma$. Therefore, we do not simulate the second part of the run in that subtree, which explains why we use a state of the form $(\sharp, q)$, using the "$\sharp$" symbol on the left so as to avoid switching to symbols of the form $\{\mathbb{0}\} \times \Sigma \times \{\mathbb{0}\}$.

Proposition 19. *Given two k-interval bounded root preserving queries Q_1 and Q_2 with $\mathrm{dom}(Q_1) = \mathrm{dom}(Q_2) = D$, $Q_1 \preccurlyeq_3 Q_2$ iff morphisms ϕ_1 and ϕ_2 are equal over $V_{2\to1}(D) \boxtimes V_{2\to1}(D)$, i.e., iff $\forall t \in V_{2\to1}(D) \boxtimes V_{2\to1}(D)$, $\phi_1(t) = \phi_2(t)$.*

Finally, we use Plandowski's result [26] stating that equivalence of morphisms on a context-free language is decidable in polynomial time. Using this result for morphisms ϕ_1 and ϕ_2 on $L(\mathcal{B}_{align})$ we get an algorithm that works in exponential time that concludes the proof of Theorem 9.